AFIPS CONFERENCE PROCEEDINGS

VOLUME 43

1974 NATIONAL COMPUTER CONFERENCE AND EXPOSITION

May 6-10, 1974
Chicago, Illinois

AFIPS PRESS
210 SUMMIT AVENUE
MONTVALE, NEW JERSEY 07645

Library of Congress Catalog Card Number 55-44701
AFIPS PRESS
210 Summit Avenue
Montvale, New Jersey 07645

Printed in the United States of America

CONTENTS

PREFACE

The 74 NCC has the objective of providing a forum for computer specialists, users, potential users and manufacturers to exchange information on new developments in computer science and technology and their applications in various areas.

The Conference Program focuses on five broad computer science and technology areas, and ten applications areas. Science and technology areas cover the latest developments which should enable users to make more efficient use of their resources in a wide range of areas. The applications areas not only demonstrate how computer technology can indeed be used to improve the efficiency of current operations, but also point out potential usage in many other areas which should generate new demands for further development in computer science and technology. Although each area forms a "conference within a conference," all fifteen areas form a comprehensive and cohesive program, instead of disjointed subprograms.

The Conference also features major addresses, a number of sessions of special interest and other activities ranging from a computer art fair, a science fair, a science theatre, to tours and an "Oklahoma Night" reception.

You'll find there are about 250 companies and organizations from the computer industry participating in the exhibits and occupying about 85,000 square feet. In spite of the size of the Conference and Exhibits, we have been able to arrange all of the Conference Program and Exhibition at one place—McCormick Place and the adjacent McCormick Inn. This arrangement should greatly help you receive the full benefits of both the Conference Program and the Exhibits.

The 74 NCC is the result of the enthusiasm, talents and efforts of many individuals from all the segments of the information processing community. In particular, I would like to thank the members of the 74 NCC Committees and AFIPS staff for their devoted efforts. A great deal of credit, however, should go to the session organizers, session chairmen, speakers, panelists and reviewers. Finally, I would like to express my appreciation to AFIPS, the NCC Board and the NCC Committee for their full support during the organization of the Conference.

Stephen S. Yau—*General Chairman*
Northwestern University
Evanston, Illinois

FOREWORD

The Conference Program has been structured around the theme of computer productivity as it relates to the user. It represents a comprehensive analysis of the U.S. computing and data processing field, covering applications of user areas, the latest developments of computer science and technology, and a number of current high interest topics relative to the impact of computers on today's society. A degree of international flavor has been added by the introduction of selected papers and panelists that can demonstrate positive and interesting differences in computer development and application in various geographical areas.

Through the concerted efforts of a committee of prominent technologists and industry specialists, the program has been developed to meet the information requirements of the corporate executive dependent on effective computer utilization, staff and line management of user organizations, management within the computing field, applications specialists and the computer professional. Each of the program areas has been shaped to form a "conference within a conference," and provides a forum where outstanding industry representatives demonstrate how recent innovations in computer technology and its application are being employed to contribute to more effective operation of their organization. In structuring the program schedule, serious consideration has been given to provide an opportunity for the attendees to benefit not only from presentations in their primary area of interest, but also to have the opportunity to be exposed to other areas of industry and technology that may be attacking similar problems.

A significant amount of time and effort on the part of a large number of interested, dedicated, professionals has gone into the development of this program. Its success, measured by your reception, is due to the participation of many volunteers who gave freely of their time and talents to organize sessions, prepare papers, referee submissions, and willingly offer advice and counsel to the Committee, who ultimately had the task of consolidating this input into the outstanding program that it is. To all who participated in the development of this program, our thanks for a job well done!

In an attempt to provide a more complete record of the Conference Program for all attendees, we have tried something new this year. In addition to publishing the formal papers, and in some cases abstracts where detailed paper material was not available at publication date, there are also included a few panel overviews and panelist position statements. Unfortunately, all panels are not represented because of the unavailability of some information at publication date.

T. M. Bellan—
Technical Program Chairman
McDonnell Douglas Corporation
St. Louis, Missouri

Some problems in associative processor applications to data base management

by P. BRUCE BERRA

Syracuse University
Syracuse, New York

INTRODUCTION

Associative memories and processors have been discussed in the literature for the past 15 years and a small number of hardware devices have actually been built.[21] The usefulness of these devices can only be proven through actual applications. A number of these applications have been considered and include air traffic control,[22] computer graphics,[24,25] information retrieval,[9] numerical analysis,[15] networks[18,19] and among others, data base management.[2,4-8,10,11,13,17]

Vast computer resources are required for the managing of large data bases. With hardware costs coming down, and software and personnel costs going up it is important that one investigate the application of associative devices to the field of data base management to ascertain what gains might be made.

In this paper, some background is given on the application of these devices to data base management. This is followed by a review of existing literature in the field. Searching, a most important capability of an associative device is then considered. It is shown that several data base management functions such as retrieval and update have searching at their core and therefore are well suited to these devices. Furthermore, due to the simpleness of the associative storage structure, increased performance can be obtained in some of the other functions of data base management and therefore one must look to these also. It is concluded that these devices have a place in the solution of data base management problems but represent only a step on the way to more sophisticated hardware/software/firmware devices designed especially for data base management problems.

DATA BASE MANAGEMENT

With the rapid growth of the computer field has come a commensurate growth in the need for software systems to manage vast amounts of numeric and non-numeric data. The development of these systems, called data base management systems, has kept pace so that there are more than 200 such systems with varying levels of capability in existence today. However, there does not seem to be any universally accepted definition for the term data base management system. But, there does seem to be some universality on the capabilities that such systems should have and some of the functions that they must perform. It seems apparent that one must go through a process of data definition in order to create a data base. This implies that there is a data structure that somehow represents the logical relationships among the data and some storage structure that is utilized in the actual storing of the data on physical media. One is given a free rein in the selection of a data structure and it often appears naturally in the problem. However, the storage structure is yet another matter. One is limited to either a sequential or a random storage structure. If one is fortunate there can be a one-to-one mapping between the data structure and the storage structure. However, this is generally not the case and redundancies must then be incurred.

Once the data base has been loaded one wants to formulate queries and extract data from it. This implies some sort of processing capability whether it be through a high level language such as COBOL or PL/1 or through some self contained capability.

Intermixed with the above is a consideration of the questions of how much storage is being used to store the data and how much is allocated to non-data such as directories, pointers, etc. Also, one must not leave out consideration for updating the data base by adding new pieces of data and deleting or changing old ones. Beyond this, such ill defined terms as flexibility, ability to respond to changing requirements, data independence, data administrator and others are often used. But this offers no difficulty here since most of the work in the application of associative devices to this field has been concerned with the more well defined aspects of data base management.

ASSOCIATIVE MEMORIES/PROCESSORS

Several of the papers referred to in this paper[5-7,14,17,21] contain background material on associative memories and processors so only limited background will be provided here. The interested reader can refer to the cited papers.

Essentially, associative memories address words in storage by content and can perform several different parallel search

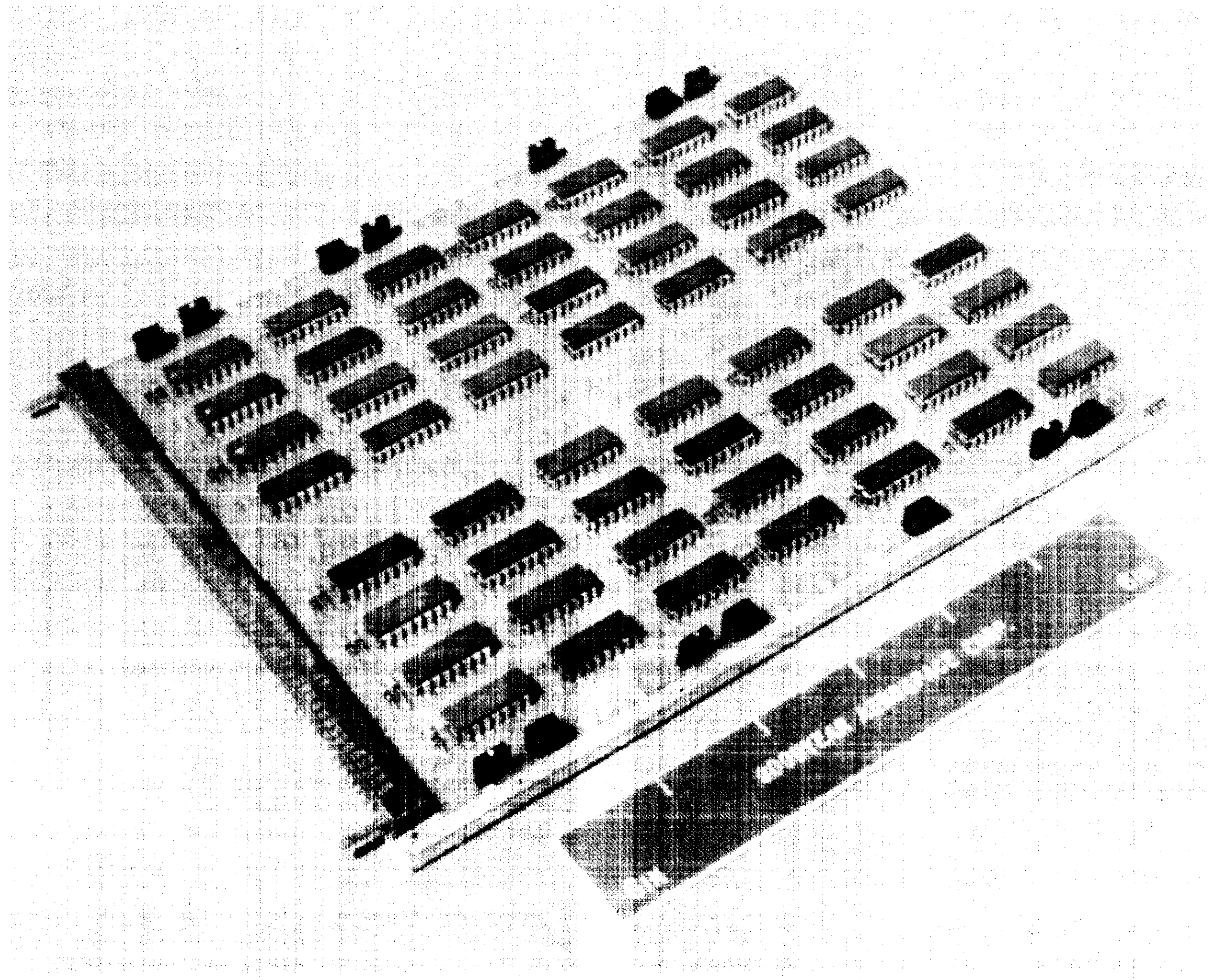


Figure 6—Typical printed circuit assembly from STARAN associative array

network and uses about the same fixed amount of logic as the processing element; e.g., 1/8 to 1/32 gates per bit of storage. The memory and the flip network together are called the Multi-dimensional Array (MDA) memory.

MDA associative array

Figure 5 shows the organization of a 256×256 associative array based on the MDA memory. The nondestructive storage function is provided by the memory, which consists of 256 standard LSI memory chips, each 256×1 bits. Access to word or bit slices is provided in the flip network, which consists of standard MSI logic circuits. The logic necessary for the associative and arithmetic functions is contained in the serial processing elements, which are implemented with standard SSI logic circuits.

The entire associative array based on a serial by bit architecture uses about 2-1/2 gates per bit compared to 40 gates per bit for the competing parallel by word and bit custom LSI approach. The complete associative array is constructed using standard printed circuit assemblies such as the one shown in Figure 6. Figure 7 shows how the associative array is viewed by the programmer. Table II gives the performance data for the 256×256 array. This associative array forms the basis for the STARAN associative processor.

TABLE II—STARAN Associative Array Performance

Multi-Dimensional Access (Bit Slice or Word Slice)	
Array Module Speed:	
Typical Search	150 Nanoseconds/Bit
Typical Add or Subtract	800 Nanoseconds/Bit
Read Bit or Word Slice (256 Bits)	150 Nanoseconds
Write Bit or Word Slice (256 Bits)	300 Nanoseconds

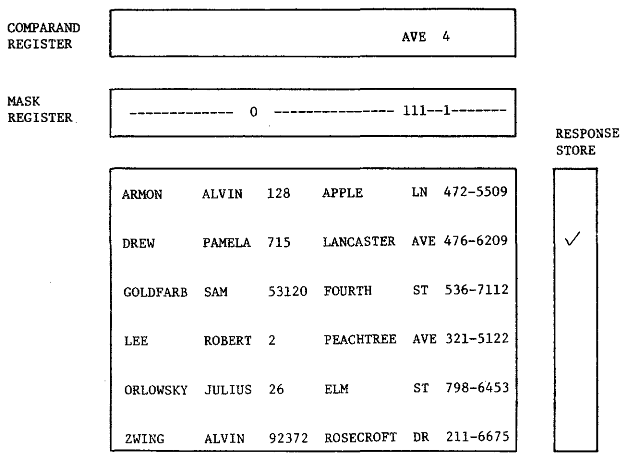

COMPARAND REGISTER
AVE 4
MASK REGISTER
------------ 0 -------------- 111--1-------
RESPONSE STORE
ARMON ALVIN 128 APPLE LN 472-5509
DREW PAMELA 715 LANCASTER AVE 476-6209 ✓
GOLDFARB SAM 53120 FOURTH ST 536-7112
LEE ROBERT 2 PEACHTREE AVE 321-5122
ORLOWSKY JULIUS 26 ELM ST 798-6453
ZWING ALVIN 92372 ROSECROFT DR 211-6675

Figure 1—Associative memory telephone example

instructions such as exact match, maximum, minimum, and others, plus Boolean operations. Associative processors can be looked upon as associative memories with arithmetic capabilities such as add, subtract, multiply or divide. Present implementations of these devices operate in a bit slice mode. That is, the operations are performed in parallel on one bit from each word. By processing each bit slice in succession the entire contents of the memory can be processed. To attain full parallelism one would have to construct the machine so that every bit position of every word would be processed simultaneously. For a comparison of various architectures see Shore.[23]

Perhaps an example, albeit trivial, will be useful in illustrating the concepts of an associative memory in searching a data base. Suppose the data base is loaded in memory as shown in Figure 1, and we would like to search for the records of those persons who live on AVENUES and have a 4 as the first digit of their telephone number. The Comparand Register is first loaded with AVE and 4 in the proper positions. The Mask Register is then loaded with ones in the position of interest and zeros otherwise. This has the effect of masking out unwanted positions in memory. An exact match search is then performed which results in a mark in the Response Store indicating that Pamela Drew's record satisfies the query. The record can then be removed for further processing if required. This simple example also serves to illustrate the mapping between relational data base management systems and associative hardware devices.

SOME RECENT RESULTS

There is a vast wealth[1] of information on data base management in a sequential computer environment but a dearth when an associative resource has been considered. Notable research that has been conducted in data base management and associative devices includes work by Moulder,[17] Linde, Gates and Peng,[13] DeFiore and Berra[2,4-7] and Goti.[10] In all of the above, simplifying assumptions had to be made for one reason or another and therefore no generalized data base management system utilizing an associative resource exists today.

In previous research by DeFiore and Berra[2,4-7] mathematical models were developed for sequential inverted list and associative systems utilizing the criteria of retrieval, update, storage requirements, and flexibility. The critical assumption of all data fitting into main memory (both sequential and associative) was made and thus limited the generality of the results.

In the case of retrieval and update, equations were developed that could be used to count the number of interrogations to main memory. The equations for retrieval were more complex than those for update since multi-criteria retrievals were considered while only single criteria updates were considered. The equations for the associative system were less complex than those for the inverted list system owing primarily to the search capability of the associative system. In the case of storage, equations for the number of bits used to store data and any redundancies were developed. This analysis yielded six equations, two each for retrieval, update and storage. Since the pairs of equations had the same units ratios were taken. This resulted in equations for retrieval and update that gave the ratio of the number of interrogations to memory for the sequential system to the number of interrogations to memory for the associative system. The equation for storage was formed in a similar way. It was felt that some numerical results would be important. In order to facilitate this some additional assumptions had to be made, primarily to remove summation terms. A discussion of the development of these equations and the resulting numerical data can be found in Reference 6.

The general results are given as follows. For single criteria searches the ratio of the number of interrogations to memory for the sequential system to the associative system was proportional to the logarithm of the list length being searched, and for multi-criteria searches ratios of 50 to 1 were common. Also, the ratio for updating was about 30 to 1 for updating a single item in a list of 16. However, the ratio was only about 3 to 1 for updating all items in the list, indicating the attractiveness of the sequential batch updating. The amount of storage required for the sequential system was from 2 to 4.5 times as much as for the associative system. Flexibility is rather difficult to define but in this work it concerned the amount of indexing that was available in the associative system versus what could be made available in the sequential system. It also concerned the ease with which one could move from one relation to another in the associative system or up and down a hierarchy in a sequential system. Because of the ease of mapping between the logical data structure and the storage structure; and the search capabilities of the associative device, it was concluded that the associative system possessed greater flexibility.

Also included in this work was the implementation of a system on an existing associative memory and the comparison of the implemented system with an existing sequential data base management system.[5] This helped to verify the general equations that were discussed above.

Moulder[17] has developed an associative system that is in operation on an experimental basis utilizing the STARAN Associative Array Processor.[21] The hardware includes a small sequential computer and a parallel head/track disk that are connected to the STARAN through a custom input/output unit. The sample data base being utilized in tests is hierarchical in nature and resides on the disk. The data base management system software includes data definition, file creation, interrogation and update modules. A search is performed in the following way. The STARAN memory is loaded from the disk in approximately 100 μsec. and searched in another 100 μsec. Then the next load of data is brought in and searched. Proceeding in this manner the entire data base can be searched in approximately two revolutions of the disk through this interleaving of loading and searching operations.

In a somewhat more general way, Goti[10] considered a system with a large data base and investigated ways of dividing the data base into memory loads so that the desired data could be retrieved while minimizing the average number of memory loads searched. Again the assumption is made that the data are in fixed format and all data in a record must be brought into the memory. However, this work allows for the development of a directory that can be utilized in the selection of which memory loads to search, whether the memory loads contain full records or keys to records.

Linde, Gates and Peng[13] developed a hypothetical Associative Processor Computer System (APCS) in order to investigate real time data management system functions. As opposed to the work by DeFiore and Moulder who used bit slice machines, the above authors suggested a byte slice machine. They investigated search, retrieval and update since they found that these are the functions most able to use an associative memory with advantage.

They normalized the APCS to the IBM 370/145 computer and found that the performance improvements varied from 32 to 110 times faster for search and retrieval and from 15 to 210 times faster for update. For loading the associative memory, their design included either a ½ or a 4 million byte mass storage device with a parallel I/O bandwidth of 1.6 billion bytes/second, depending upon the function being considered.

For the past few years, research has been carried out in distributed logic systems. Some of this work has been directed toward placing logic elements on rotating storage devices. These devices can be designed to have the same search capabilities as the associative memory. This work can be looked upon as a lower cost solution to the associative memory size problem but at the expense of slower speed.[3,12,20]

To this author's knowledge, there is little else in the literature concerning data base management and associative processing with the exception of some early work by Green, Minker and Shindle[11] and some recent work by Downs[8] with Illiac IV.

SOME ADVANTAGES OF ASSOCIATIVE DEVICES

Having reviewed the existing literature concerning the application of associative memories and processors to data base management one can ask about the usefulness of these devices for this type of problem. Looking to the work of previous authors we see that the functions of search, retrieval, and update have been considered. The amount of computer storage utilized for the data and any redundancies has also been considered. Finally, the flexibility of a system has been considered.

Ignoring parallel arithmetic operations for a moment, the operation that associative devices can offer to the solution of data base management problems is rapid search of data in memory. And search operations are at the heart of such data base management functions as retrieval (both single and multi-criteria), update, merging and sorting.

Although one cannot directly attribute an effect of searching to the amount of storage utilized there is nevertheless an indirect effect. For instance, we learned from Reference 6 that the amount of storage required in main memory was from 2 to 4.5 times greater for the sequential computer than for the associative device primarily because directories were not needed in the latter case. Although the above referenced work was carried out under the assumption of all data fitting into memory, one can extrapolate to a data base consisting of a great many memory loads. Thus, this would represent a considerable saving in the amount of storage required for the data base as well as having much less data to search.

Although flexibility is difficult to quantify, the fact that each bit or any combination of bits of a word can be used as a key for searching indicates that flexibility is increased for the associative device, at least in those operations that have been considered in the literature so far.

Another possible indirect advantage of these devices is in the software. Although the data are sparse at this point, it appears that the programming of these devices is considerably simplified both in logic and in the compactness of code.

In studying the use of associative memories and processors in data base management, it has become clear that the arithmetic capabilities of the associative processor are seldom required in data base problems since the opportunity for the parallel processing of large quantities of data rarely exists. So, at the present time it seems that it would only be useful to utilize associative memories in this field. One exception to this concerns mass updating of the data base in a real time environment. For instance, in air traffic control applications the data base of tracks may have to be manipulated in real time and the need may arise to update all tracks simultaneously. But this kind of mass updating would seldom be required for a business or industrial type data base management problem.

SOME DISADVANTAGES OF ASSOCIATIVE DEVICES

The application of associative memories and processors to data base management is not without its own share of problems. The associative memory must be loaded before any searching can take place and this may not be an easy task. From a technological point of view the problem has essentially been solved but it may take a sizable amount of high speed storage to keep the associative device rapidly supplied with data for processing. As indicated earlier, in work by Moulder[17] a head/track disk is used to supply the associative processor with data. After the memory has been loaded it may take only 100 μsec. to search and retrieve the necessary data before the memory is ready for loading again. Also indicated earlier was work by Linde, Gates and Peng[13] in which a ½ or 4 million byte mass storage device with a parallel I/O bandwidth of 1.6 billion bytes/second was assumed.

Another problem that exists is that of fixed field formatting. As can be seen in Figure 1, the data must be left or right justified in order to exploit the parallel search capability of the memory. This means that the same number of bits must be allocated to the same data items in each record (i.e., 10 character positions for Su or Pennacchia). This may be wasteful of storage but must be done in order to allow rapid search. A possible solution to this problem is the use of delineators but this creates some additional problems that may well degrade the performance of the system.

Still another problem that presently exists is the current size of the memory. Sequential computers have been mass produced for many years now but associative devices are in the one of a kind stage. Because of this they are necessarily small and costly. However, as more problems are solved using these devices, the sizes will increase and the costs will be reduced. With sequential computers there is always a demand for a larger machine than is presently available and this should not be any different with associative devices. In order to help alleviate the problem there may well be a virtual memory philosophy applied to associative processors but this is probably still a few years off.

FUTURE RESEARCH

One of the messages of this paper is that it appears at the present time that associative memories and processors have a potential for reducing some of the pressing problems in the field but that they are by no means the final answer; only a step on the way to more sophisticated devices. It has long been the contention of this author that there are thousands of data base problems in existence today that would support the development of computers strictly for the solution to these problems. Imagine the vast amounts of data the various government agencies must manage, let alone all of the industrial organizations and businesses that are aspiring to integrated corporate data bases. Imagine also the vast amount of computer resources that are wasted in processing largely non-sequential data on sequential computers.

Now what can be said of the future? It seems clear to this author that for the next few years the associative device will remain essentially a peripheral to a sequential computer. One reason is that we just don't know enough about the generic functions that must be performed in data base management and therefore can't really define what we need from the hardware. Another reason is that we are used to thinking in terms of the sequential computer. Hopefully this will change but it will be a slow and sometimes painful process. One need only witness the behavior of those who have attempted to make the transition to the parallel field with a bagful of good sequential algorithms only to find out that some poor obscure sequential algorithms worked more efficiently in the parallel environment than the ones in his/her bag.

The research that has been reported on in this paper has considered only limited sized data bases largely because of the small size of the associative device and the I/O problem. But, what of the large data base problem? The work by Goti[10] is important in this area. Using mathematical models his approach was that of partitioning the data base into blocks that could be made approximately equal in size to an associative resource. This could then facilitate the processing of the records once they were found. The data base was assumed to be on conventional sequential storage media. Although his work is independent of whether one uses an associative resource or a sequential resource the processing of some sort of directory to the data base can be enhanced through the use of the associative resource. This leads to the consideration of what kind of directory to build when one has the resource available. We are presently considering this problem but results are not available as yet.

During the next few years work will continue in the placement of logic on rotating devices in order to obtain the same search capabilities as the associative memory. The added advantage is mostly in terms of cost but the fixed format and I/O problems are reduced considerably. The size limitation still exists and speed is of course slow in comparison to an associative device but for some applications this presents no problem.

In the opinion of this author, the real gains will come when we are able to provide a one-to-one mapping between the data structure and the storage structure for a wide class of data structures. Some information is available already in that the data structures for some relational data base management systems have more of a one-to-one mapping with associative devices than with sequential devices. But, to really address the area we need to study large numbers of data base management problems independently of any existing systems in order to define a set of generic functions. We can then select the most efficient implementations of these functions whether in hardware, firmware, software or combinations of the three. This will take a great deal of time and effort but will be extremely important to the efficient solutions of data base management problems.

REFERENCES

1. Berra, P. B., *Some References in Data Management*, May, 1973, available from the author on request. (Approximately 200 references.)
2. Berra, P. B., "A Synopsis of Research Results in the Applications of Associative/Parallel Processors to Operations Research, Data Management and Change Detection," *1972 Sagamore Computer Conference Proceedings*, Syracuse University, August 23-25, 1972.

3. Copeland Jr., G. P., G. J. Lipovski and S. Y. W. Su, "The Architecture of CASSM: A Cellular System for Non-numeric Processing," *Proceedings of the 1st Annual Symposium on Computer Architecture*, Dec. 9-11, 1973.
4. DeFiore, C. R., *An Associative Approach to Data Management*, unpublished doctoral dissertation, Syracuse University, May, 1972. (Also as RADC-TR-72-248, September, 1972.)
5. DeFiore, C. R. and P. B. Berra, "A Data Management System Utilizing an Associative Memory," *AFIPS Conference Proceedings*, Vol. 42, June, 1973.
6. DeFiore, C. R. and P. B. Berra, "A Quantitative Analysis of the Utilizations of Associative Memories in Data Management," *IEEE Transactions on Electronic Computers*, February, 1974.
7. DeFiore, C. R., N. J. Stillman, and P. B. Berra, "Associative Techniques in the Solution of Data Management Problems," *Proceedings 1971 ACM National Conference.*
8. Downs, H. G., "Real-Time Algorithms and Data Management on Illiac IV," *IEEE Transactions on Electronic Computers*, Vol. C-22, No. 8, August, 1973.
9. Goodyear Aerospace Corporation, "The Application of an Associative Memory to Chemical Information Storage and Retrieval," *GER 13224*, April 14, 1967.
10. Goti, J. C., *Optimal n-Rectangular Partitioning of Large Data Bases for Multiple Attribute Retrieval*, unpublished doctoral dissertation, Syracuse University, November, 1973.
11. Green, R. S., J. Minker, and W. E. Shindle, "Analysis of Small Associative Memories for Data Storage and Retrieval Systems," Vols. 1 & 2, *RADC-TR-65-397*, July, 1966.
12. Healy, L. D., G. J. Lipovski and K. L. Doty, "The Architecture of a Context Addressed Segment-Sequential Storage," *AFIPS Conference Proceedings*, Vol. 41, December, 1972.
13. Linde, R. R., R. Gates, and T. Peng, "Associative Processor Applications to Real-Time Data Management," *AFIPS Conference Proceedings*, Vol. 42, June, 1973.
14. Minker, J., "Bibliography 25: An Overview of Associative or Content-Addressable Memory Systems and a KWIC Index to the Literature: 1958-1970," University of Maryland and Auerbach Corporation, *Computing Reviews*, October, 1971.
15. Miranker, W. L., "A Survey of Parallelism in Numerical Analysis," *SIAM Review*, Vol. 13, No. 4, October, 1971.
16. Minsky, N., "Rotating Storage Devices as Partially Associative Memories," *AFIPS Conference Proceedings*, Vol. 41, 1972.
17. Moulder, R., "An Implementation of a Data Management System on an Associative Processor," *AFIPS Conference Proceedings*, Vol. 42, June, 1973.
18. Orlando, V. A., *Associative Processing in the Solution of Network Problems*, unpublished doctoral dissertation, Syracuse University, February, 1972.
19. Orlando, V. A. and P. B. Berra, "The Solution of the Minimum Cost Flow and Maximum Cost Flow Network Problems Using Associative Processing," *AFIPS Conference Proceedings*, Vol. 41, 1972.
20. Parhami, B., "A Highly Parallel Computing System for Information Retrieval," *AFIPS Conference Proceedings*, Vol. 41, December, 1972.
21. Rudolph, J. A., "A Production Implementation of an Associative Array Processor-STARAN," *AFIPS Conference Proceedings*, Vol. 41, 1972.
22. Rudolph, J. A., L. C. Fulmer, and W. C. Meilander, "The Coming of Age of the Associative Processor," *Electronics*, February, 1971.
23. Shore, J. E., "Second Thoughts on Parallel Processing," *NRL Report 7364*, Naval Research Laboratory, Washington, D. C., December 30, 1971.
24. Stillman, N. J., *A Feasibility Study of the Applicability of a Hardware Associative Memory to Computer Graphics*, unpublished doctoral dissertation, Syracuse University, February, 1972. (Also as *RADC-TR-72-57*, April, 1972.)
25. Stillman, N. J., C. R. DeFiore, and P. B. Berra, "Associative Processing of Line Drawings," *AFIPS Conference Proceedings*, Vol. 38, 1971.

RADCAP—An operational parallel processing facility

by JAMES D. FELDMAN and LOUIS C. FULMER

Goodyear Aerospace Corporation
Akron, Ohio

SUMMARY

An overview is presented of RADCAP, the operational associative array processor (AP) facility installed at Rome Air Development Center (RADC). Basically, this facility consists of a Goodyear Aerospace STARAN* associative array (parallel) processor and various peripheral devices, all interfaced with a Honeywell Information Systems (HIS) 645 sequential computer, which runs under the Multics time-shared operating system. The RADCAP hardware and software are described only briefly here since they are detailed in companion papers presented at this conference.[1,2] The latter part of this paper dwells on the objectives of the RADCAP facility and plans for its use.

The STARAN associative parallel processor is a processor based on an associative or content addressable memory and a related ensemble of bit serial processing elements. STARAN is considered to be the first practical associative processor ever produced.[3] This claim of practicality is based on the fact that the design concept for the associative memory of STARAN allows the use of the same high-volume, standard, large-scale integrated (LSI) circuit memory devices that are in widespread use by the computer industry. In fact, every electronic component used in the STARAN associative parallel processor is available from your local components distributor. The significance of this fact is that now, for the first time, associative processors enjoy the same cost per bit of storage as does the conventional computer.

HISTORICAL BACKGROUND

From the time Slade and McMahon first described their catalog memory[4] in 1957, many attempts have been made to implement an associative memory. Some of these attempts were successful, but until recently none has been very practical. Table I lists some of the device technologies that have been used in the past to implement associative memories in the laboratories and in a few experimental models. Except for a few special applications of plated wire, none of these device technologies have ever been used successfully for conventional memory technology. Further, for associative memory applications, none of these device technologies have been very practical. All had one common characteristic—high cost per bit of storage. The exotic nature of the device, the custom nature of the associative cell, and the resulting low volumes were the factors contributing to the high cost per bit.

The extension of an associative memory to an associative processor by the addition of serial arithmetic units to each word of associative storage was demonstrated using plated-wire technology and was reported[5] in 1970. At about that same time a new version of the STARAN associative parallel processor was in the formative design stages at Goodyear Aerospace Corporation. Two choices for the device technology of the associative array were available: plated wire or LSI. Plated wire was quickly discarded as a "squeezed" memory technology caught between the well established magnetic core memory and the emerging solid-state integrated circuit memories. The choice was to go with LSI, and the temptation to design a custom LSI associative array was great.

The design of the associative array could be made parallel by word and bit and could include a repetitive cellular structure that would lend itself nicely to an LSI array. The cell could be designed to include all of the desirable characteristics of an associative processor: nondestructive readout storage, logic for the associative and arithmetic functions, and access to the cell in both the word and bit slice directions.

Figure 1 is a simplified diagram of a two-dimensional custom LSI associative array, where each cell has been designed to include the desirable storage, logic, and access characteristics of an associative processor. The largest neg-

TABLE I—Associative Memory Element Technologies

Cryogenics
Cryoelectrics
Multiaperture Ferrites (MADS, MALE, BIAX)
Ferroelectrics
Toroidal Cores (FLUXLOK, BILOC)
Discrete Transistor Associative Cells
Discrete Integrated Circuit Associative Cells
Plated Wire
Custom LSI

* TM, Goodyear Aerospace Corporation, Akron, Ohio.

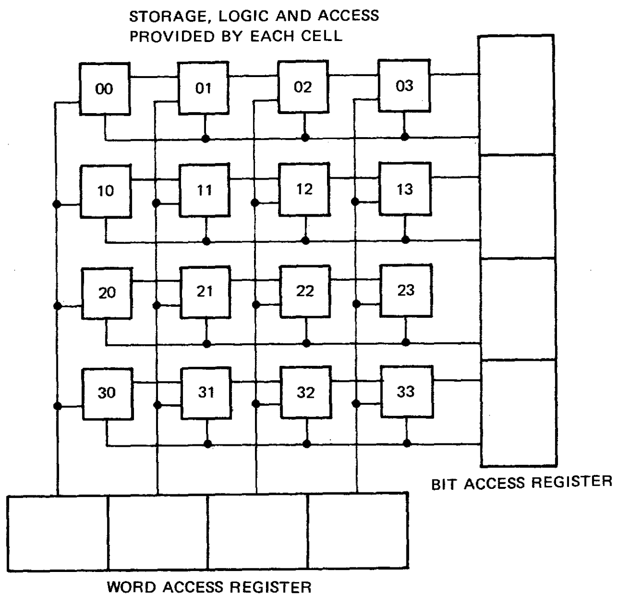


Figure 1—Custom LSI associative memory (parallel by word and bit)

ative technical factor in this approach is the large number of gates for each associative cell. In one design[6] of this type, the associative cell required about 40 gates. The high gate count implies more silicon per cell and, therefore, lower yield and higher cost. A second technical factor is the large number of pins required to package the two-dimensional LSI array. Another design[7] that packages 128 associative cells requires a 40-pin package and 256 cells requires a 56-pin package. Tradeoffs between pin count and logic complexity are possible but limited. The high pin count results in the use of expensive nonstandard integrated circuit packages. A third technical factor in a design of this type is that heat dissipation, geometry, and economics dictate the use of external sense amplifiers.

Even if all of the above technical problems could be solved economically, the largest problem is a nontechnical one—that of low volume. Due to the custom nature of the device, it can only be used in the associative processor for which it is designed, and unless an integrated circuit is produced in response to a tremendous volume demand, its cost will always be relatively high. Related to this custom device problem is the fact that the designer of the associative processor must bear the high nonrecurring costs by himself, usually with a single source which further insures relatively high prices, and without the benefit of a background of reliable data for the specific device. Then, due to the high nonrecurring investment and the low volumes, the associative processor designer will not be able easily to take advantage of the technological advances that are occurring rapidly in the integrated circuit industry and he soon has an obsolete device on his hands.

Further, the integrated circuit industry does not want to be involved in low volume production programs. An excerpt from a 1973 report for the government entitled "Approaches to Custom Large Scale Integration"[8] is quoted below to support this point.

"It is increasingly apparent that LSI offers considerable benefits, most of them related to cost, in the implementation of the digital portion of any system of reasonable complexity, provided it is produced in high volume. High volume can be taken for granted in standard LSI devices, such as memories. Such is not the case for logic, however, which, in order to be near optimum, must be custom for each function; the volume in which it is produced therefore depends entirely on the number of systems to be built. While the total number of systems required by the military may be large, the number of any one system type is often small. However, in order to reap the full cost benefits of LSI, the volume must be high, at least of the order of $100,000 to $200,000 per year per chip, and preferably higher. Only at this level does it become economically feasible to fully optimize the design by handcrafting for maximum area utilization and performance, to "tweak" the process for maximum yield, and to package at the lowest cost. These cost savings are not realized at lower volume.

The following summarizes the major comments made by the industry:

- *High volume is the most important business criterion.*
- *Low-volume LSI development work was done principally with military funding. Work stopped when funding dried up.*
- *Low-volume LSI is bad business for the big semiconductor companies.*
- *Only systems houses with captive IC capability can respond to low-volume LSI requirements.*
- *Manufacturing in IC houses is geared to very large lots.*
- *The customer must be very sophisticated if he wants low-volume LSI, preferably he generates his own design.*
- *Limiting resources are: 1. Capital equipment; 2. Manpower. These cannot be wasted on low-volume businesses.*
- *Every effort should be made to use standard products.*

The custom LSI approach was considered for STARAN but was discarded primarily due to the nontechnical factors discussed above. A custom LSI approach, although technically appealing, would result in an impractical implementation of the associative processor. That approach can never break out of the "cost-volume hangup."

The GAC approach to using LSI for the associative processor was just the opposite of the custom LSI approach. Instead of combining the requirements of storage, logic, and access into one custom LSI chip, the requirements were divided to see if they could be implemented using several standard, high-volume large-, medium-, and small-scale integrated circuit chips. Figure 2 is a schematic representation of the approach to the problem. If this problem could be solved, the resulting cost of associative processors would be low and the "cost/volume hangup" would be solved.

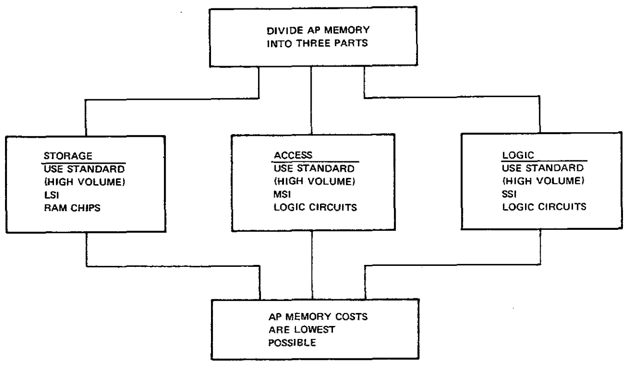

DIVIDE AP MEMORY INTO THREE PARTS
STORAGE
USE STANDARD (HIGH VOLUME) LSI RAM CHIPS
ACCESS
USE STANDARD (HIGH VOLUME) MSI LOGIC CIRCUITS
LOGIC
USE STANDARD (HIGH VOLUME) SSI LOGIC CIRCUITS
AP MEMORY COSTS ARE LOWEST POSSIBLE

Figure 2—Approach to implementing an associative processor memory

Storage

Figure 3 shows a typical standard LSI 1024×1 memory chip in a 16-pin package. These memories are (or soon will be) available in high-volume production in various device technologies (MOS, TTL, ECL, CMOS) and are in widespread use in the conventional computer industry for mainframe memory. Since they are committed to replace the magnetic core being used for that purpose, the volume leverage for lower cost is already at work. For use in a serial by bit (slice) associative processor, this memory chip is functionally equivalent to single plated wire and can satisfy the nondestructive storage requirement very nicely. The number of gates per bit of storage is approximately two.

Logic

In a serial by bit associative processor, the logic necessary to perform the associative and arithmetic functions is embodied in a small bit-serial processing element sometimes called a response store or a serial arithmetic unit. Typically, this processing element has a complexity of about 32 gates and consists of three or four flip-flops and some logic gates.

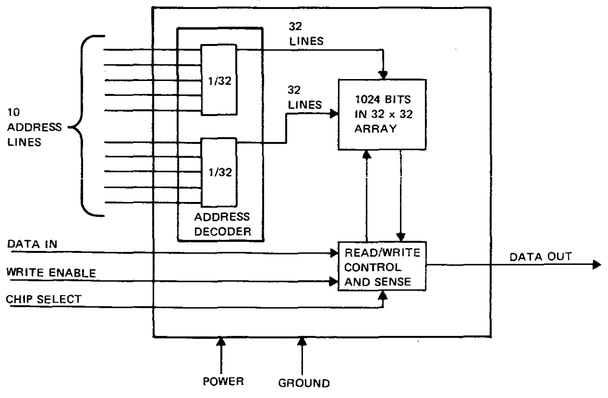


Figure 3—Standard LSI random access memory (1024-bit chip in 16-pin package)

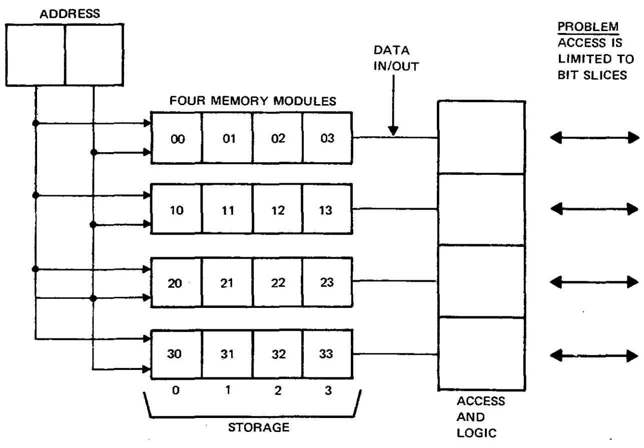


Figure 4—Standard LSI memory used in an associative processor

It is easily constructed of small-scale integrated (SSI) circuits. When operated with a 256-bit store, the processing element adds 32/256 or 1/8 gate per bit of storage. With a 1024-bit store, the ratio is 32/1024 or 1/32 gate per bit of storage.

Access

Figure 4 shows a simplified diagram of four LSI memory chips connected to four serial processing elements. This organization is very similar to the plated-wire serial by bit associative processor mentioned earlier. The problem with this design is that access to a bit slice is accomplished in one-bit read or write time but access to a word or part of a word requires (n) bit read or write times (where n is the number of bits to be read or written). It would be desirable to access either a bit slice or a word slice in one read or write time. There may also be cases where it is desirable to access in a mixed mode (words and bits) addressing technique that allows up to 256 cells to be accessed at once.

The problem of access to either bit slices or word slices or combinations of the two has been solved with a proprietary GAC design that uses a logical network between the memory and the processing elements. This network is called a flip

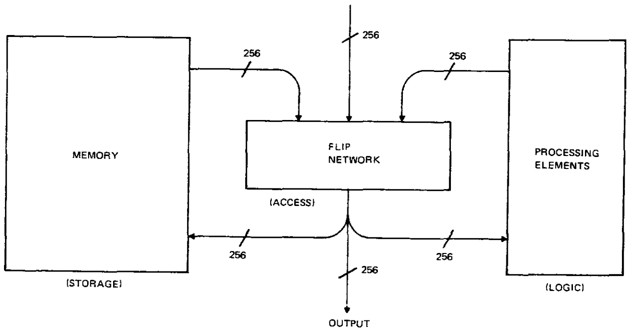


Figure 5—Organization of a STARAN associative array

RADCAP FACILITY

Figure 8 shows a block diagram of the hardware within the RADCAP facility. The 645, which has been in existence at RADC for several years, is a very large computer system with a multitude of peripherals typical of large time-shared systems. In March 1973, hardware was delivered to RADC in the form of a STARAN parallel processor with four arrays, a custom input/output unit (CIOU), a hardware performance monitor, and a variety of peripherals. Subsequently, the CIOU was used to interface STARAN with a 645 I/O channel. At the same time, STARAN software was interfaced with the 645 Multics time-shared operating system.

At present, the RADCAP facility is totally operational and includes system software to allow for operation in both a STARAN stand-alone mode and an integrated STARAN/Multics mode.

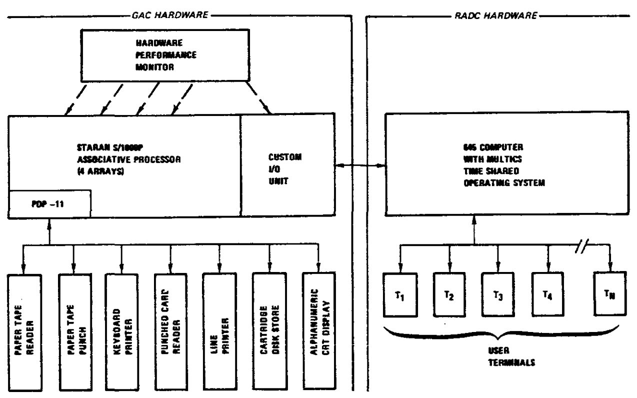


Figure 8—RADCAP facility

STARAN PARALLEL PROCESSOR

STARAN can perform search, arithmetic, and logical operations simultaneously on either all or selected words of its memory. Figure 9 shows the basic STARAN elements. The most important are the associative array and its unique multi-dimensional access capability, which, along with the other elements, are described in more detail in referenced publications.[1,3,9] Listed below are brief descriptions of the STARAN elements:

1. Associative array: provides multi-dimensional access, content-addressable memory with 65,536 (2^{16}) bits of storage and 256 processing elements; permits parallel arithmetic, search, and logical operations.
2. AP control: performs data manipulation within associative arrays as directed by program stored in AP control memory.
3. AP control memory: stores AP control instructions. Can also store data and act as buffer between AP control and other system elements.
4. Sequential controller and memory: performs maintenance and test functions, controls peripherals, maintains job control, provides means for operator communication between various STARAN elements, and assembles STARAN programs written in MAPPLE (*M*acro-*A*ssociative *P*rocessor *P*rogramming *L*anguag*E*).
5. External functions: transfers control information among STARAN elements.

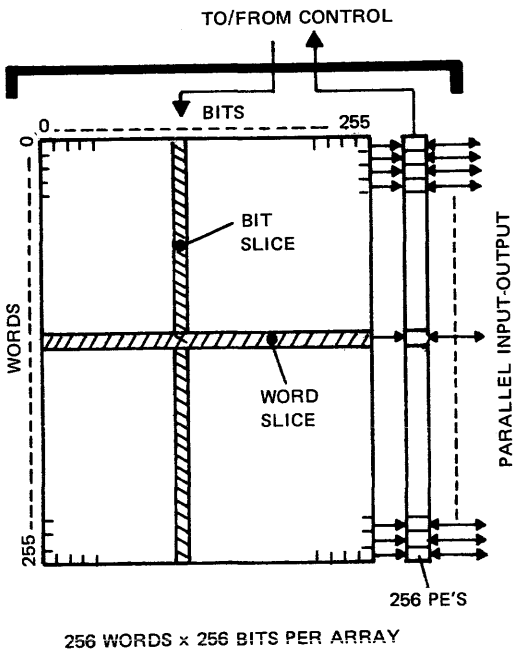


Figure 7—STARAN associative processor array

STARAN has been designed to provide a flexible I/O capability. The standard peripherals for STARAN are listed below, along with a typical list of optional peripherals:

1. Standard: cartridge disk drive and control, paper tape reader, paper tape punch, and keyboard printer.

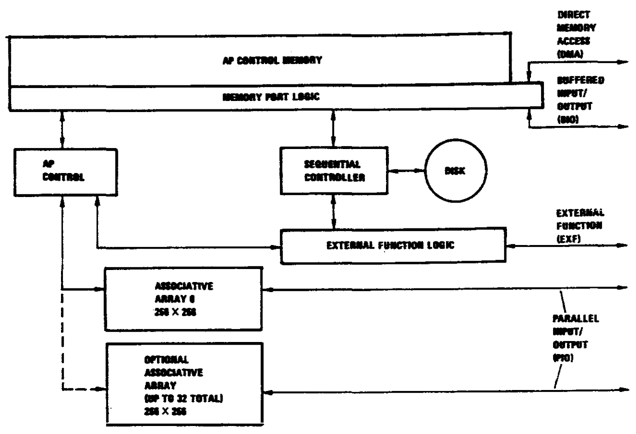

AP CONTROL MEMORY
MEMORY PORT LOGIC
DIRECT MEMORY ACCESS (DMA)
BUFFERED INPUT/OUTPUT (BIO)
AP CONTROL
SEQUENTIAL CONTROLLER
DISK
EXTERNAL FUNCTION (EXF)
EXTERNAL FUNCTION LOGIC
ASSOCIATIVE ARRAY 0 256 X 256
OPTIONAL ASSOCIATIVE ARRAY (UP TO 32 TOTAL) 256 X 256
PARALLEL INPUT/OUTPUT (PIO)

Figure 9—STARAN block diagram

2. Optional: line printer, card reader, magnetic tape, keyboard crt, and other peripherals, as desired, that are compatible with the Digital Equipment Corporation (DEC) PDP-11.

All these peripherals interface with the STARAN system's sequential controller, a PDP-11 mini-computer. STARAN also provides facilities for interfacing with other processors. The four buses provided (see STARAN block diagram, Figure 9) are the direct memory access, the buffered I/O, external function, and parallel I/O.

The direct memory access is a 32-bit bus for STARAN to address external memory. The AP control or the sequential controller can access external memory at a rate dependent upon this memory's cycle time.

The buffered I/O is a 32-bit bus for processors to address STARAN. Depending upon which portion of control memory is accessed, the access rate is 0.4 to 1.0 microsec per 32-bit word.

The external function is a bus for exchange of control signals. Discrete signals and interrupts can be both generated and accepted across this bus.

The parallel I/O is a bus for STARAN array I/O. Up to 256 bits per array (e.g., one bit per array word) can be provided. If all 32 arrays are implemented, up to 8192 bits can be utilized in parallel at a transfer rate less than one microsecond, dependent upon the desired application.

STARAN PERFORMANCE SUMMARY

In a high-speed, asynchronous, pipe-line type processor such as STARAN, it is difficult to summarize performance since speeds vary with instruction types, types of loops, etc. Also, the overall effective speed depends upon the number of words in the arrays over which the simultaneous operations are occurring. However, an effort is made below to list the performance and the features of 256×256 associative array, the control unit, and the interface portion of STARAN:

Associative Array Features

- Up to 32 arrays per system
- Multi-dimensional access (bit slice or word slice)
- Array module speed:

Typical search:	150 nsec/bit
Typical add or subtract:	800 nsec/bit
Read bit or word slice (256 bits):	150 nsec
Write bit or word slice (256 bits):	300 nsec

Control Unit Features

- Two separate processors: AP control, sequential controller
- Solid-state control memory capacity: 2K×32 standard, 4K×32 maximum
- Solid-state control memory speed: 150 nsec/instruction (typical)
- Bulk core capability: 16K×32 standard, 32K×32 maximum
- Bulk core speed: 1 microsec (read or write)

Interface Capabilities

- STARAN to address external memory: rate-memory dependent
- External processor to address STARAN: 0.4 to 1.0 microsec/32-bit word
- Parallel I/O to/from associative arrays: less than 1.0 microsec/8192 bits (maximum)
- Control signals and interrupts

Custom Input/Output Unit (CIOU)

Figure 10 shows a simplified block diagram of the STARAN/RADCAP custom input/output unit (CIOU). As indicated, the CIOU contains a parallel input/output (PIO) module, a 645 computer interface, and an internal performance monitor. The CIOU functions as a mini-processor much the same as the control unit portion of STARAN. Processing within one array module (e.g., under STARAN control) may be concurrent with I/O in another array module (e.g., under PIO control).

As directed by instructions stored in PIO control memory, the optional PIO module manipulates data among and within the associative arrays concurrent with operations as directed by AP control. The PIO module contains eight ports, with 256 bits per port to accommodate associative array I/O and to permute data.

The 645 interface logic provides a communication path between the 645 computer and the STARAN system. This interface logic contains a 30-character queue and a 32-bit status register, which are tied to a 645 I/O channel. The status register contains interface control signals, and the queue buffers data being transferred to or from the 645.

The internal performance monitor, although contained in the CIOU, is best discussed in the following description of the hardware performance monitor.

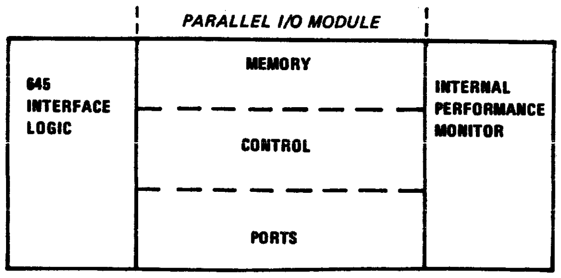


Figure 10—Simplified block diagram of custom I/O unit

Hardware performance monitor

To help meet a RADCAP facility objective of measuring system performance, a hardware performance monitoring capability has been provided by an internal performance monitor in the CIOU cabinet and an external performance monitor system. Measurements can be made to determine instruction execution timing, control memory and bus utilization, array utilization, and activity in the pager, the PIO module, and the 645 interface.

The internal performance monitor is used exclusively for STARAN instruction execution times and instruction event times. The events counted and timed are the execution of flagged instructions in AP control. Between a start flag and an end flag, a timer increments at a 100-nsec rate. Overflows from this counter interrupt the sequential controller. In addition, the sequential controller can interrogate the event counter and timer.

The external performance monitor is a self-contained system that can monitor any point of STARAN or the custom I/O. Data are acquired via probes that detect logical signal changes in either an event count or elapsed time mode. Several probes can be logically connected via a patchboard to trigger a counter. At regular intervals, the contents of the counters are written as a record on a magnetic tape unit. The performance monitor software then evaluates the collected data and produces the results in the form of reports and graphs. The software for the performance monitor runs on the 645.

Physical description of hardware

All the elements shown in the STARAN block diagram (Figure 9), including the associative arrays, are built using dual-in-line IC's (integrated circuits) mounted on multilayer printed circuit boards. Thus, the physical construction of STARAN and the CIOU is similar to that of typical high-speed sequential processors.

Figure 11 shows Goodyear Aerospace's STARAN demonstration and evaluation facility. Table III gives the approximate numbers of cabinets, boards, and IC's for the various STARAN models. These figures do not account for I/O logic, since this varies from one installation to another. The STARAN/RADCAP CIOU, which includes the parallel I/O option for all four arrays, contains approximately 200 boards and 8,000 IC's.

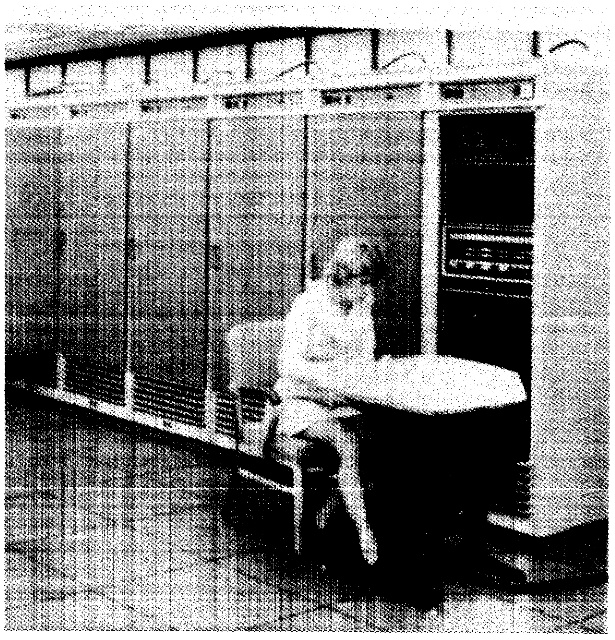


Figure 11—STARAN demonstration and evaluation facility

TABLE III—Approximate STARAN Component Count*

STARAN* Model	No. of Arrays	No. of Cabinets	No. of Printed Circuit Boards	No. of Integrated Circuits
S-250	1	3	220	9,000
S-500	2	3	276	11,500
S-750	3	3	332	14,100
S-1000	4	4	412	16,700
S-1250	5	4	468	19,300
S-1500	6	4	524	21,900
S-1750	7	5	604	24,900
S-2000	8	5	660	27,500
S-4000	16	8	1156	48,700

* Without input/output.

Although up to three arrays can be packaged in one cabinet, the RADCAP configuration has two arrays per cabinet for symmetry. Figure 12 shows the equipment that was delivered to RADC. This includes a sequential control cabinet, an AP control cabinet, two AP memory cabinets for the four associative arrays, and a CIOU cabinet. The disk drive and line printer are mounted in separate cabinets. The keyboard/printer, the card reader, and the graphics display console can be mounted on table tops or pedestals. As mentioned earlier, the internal performance monitor is packaged within the CIOU cabinet. The external perfor-

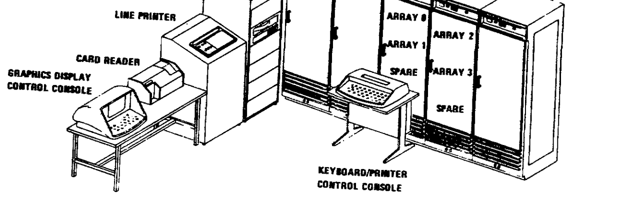


Figure 12—STARAN complex at RADC

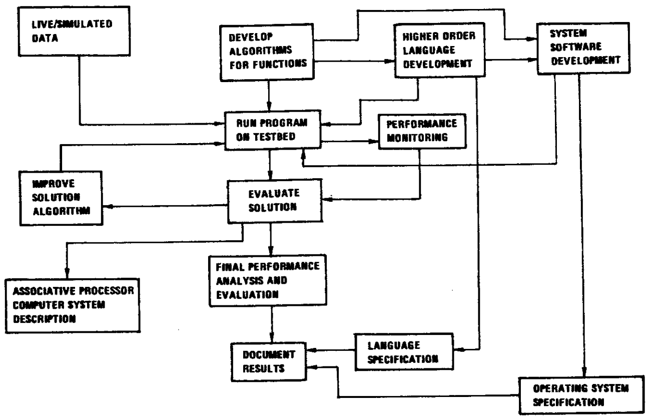

LIVE/SIMULATED DATA
DEVELOP ALGORITHMS FOR FUNCTIONS
HIGHER ORDER LANGUAGE DEVELOPMENT
SYSTEM SOFTWARE DEVELOPMENT
RUN PROGRAM ON TESTBED
PERFORMANCE MONITORING
IMPROVE SOLUTION ALGORITHM
EVALUATE SOLUTION
ASSOCIATIVE PROCESSOR COMPUTER SYSTEM DESCRIPTION
FINAL PERFORMANCE ANALYSIS AND EVALUATION
LANGUAGE SPECIFICATION
DOCUMENT RESULTS
OPERATING SYSTEM SPECIFICATION

Figure 13—Flow of RADCAP research project

mance monitor, not shown in Figure 12, mounts on a table top.

Summary of system software

The system software available for STARAN/RADCAP is capable of operating STARAN in a stand-alone mode or, when integrated with the 645, in a STARAN/Multics configuration. The system software is based upon a disk operating system, which provides ready access to system programs, device independent I/O, and a file system. Operation of STARAN can be under direct control of the user at the control console or run in a batch mode with a control stream from an input device like the card reader.

The total assembly package for STARAN has a macro-language processor, an APPLE assembler, and a relocating linker. Programs are written in the APPLE and MAPPLE languages. Extensive string handling and substitution are implemented in the macro-preprocessor. APPLE is a symbolic language that includes mnemonics for parallel and associative operations. The linker combines separately assembled object modules by relocating code as necessary and resolving globally defined symbols.

Control of processing in STARAN is through interactive system routines. These routines are the interface between application program execution and the user. They allow the user to start and halt STARAN, to load programs and overlays, and to debug programs with trace, memory modification, and dump commands.

Diagnostic programs for STARAN hardware are disk resident. The programs can be called individually, in groups related to specific parts of the hardware, or as a total set for complete system testing. Fault detection and location are provided.

Additional software for the integrated STARAN/Multics operation is designed to handle the interface between the computers and the use of STARAN from Multics. For the interface, a special device driver module has been added to the STARAN disk operating system. This driver is similar to drivers used for peripherals. It has been specialized for Multics and can accommodate 16 open files simultaneously. A device interface module (DIM) has been added to Multics as the counterpart to the device driver. These two modules are basic parts of each machine's operating system and are transparent to the programmer.

STARAN can be operated from Multics by commands a user inputs at a terminal or from a file. File control procedures handle STARAN related keyboard inputs and provide the interface between the DIM and the MULTICS storage system. With these procedures, a user process executing in the 645 can call for execution of a STARAN program.

To facilitate the assembly of STARAN programs, a cross assembler is provided for time-shared use in Multics. This assembler accepts MAPPLE and APPLE as inputs.

Objectives and uses

The basic objective of the RADCAP facility is to explore the performance of a hybrid computer configuration (STARAN associative processor interfaced with a 645 sequential processor) on real-world, real-time problems. A specific goal is to determine the cost-effectiveness of associative/parallel processing in such an environment. Associative processing has been studied extensively in both theoretical and simulation studies, but no significant practical operating experience with them exists. Experimentation is necessary to provide "hard" data and fill in the presently existing void. Practical operating experience also is required so that a general-purpose associative processor configuration could be developed if results warrant it.

The RADCAP facility will be used in an experimental program to evaluate the internal performance of this hybrid computer configuration by means of hardware and/or software performance monitors to determine internal component utilization and system bottlenecks. Programming aspects of associative processing also will be investigated. Associative-processing programming is not well understood and represents radical departures from the traditional programming approach. The program loop is being replaced by hardware processing elements. This requires a whole new programming attitude. Programming languages suitable for associative processors probably will be quite different from present ones. This basic uncertainty must be explored and some practical operating experience gained. As a test problem, indicative of high data rate and real-time processing requirements, the data processing functions of an air surveillance system (AWACS) have been chosen. The primary functions to be investigated are tracking (both passive and active), display processing, and weapons control.

The scope of the research program can be described with the aid of Figure 13. The flow will begin with the development of associative-sequential algorithms for each of the AWACS data processing functions. As these algorithms are being developed, the application engineers will make known to a language and system software group those instruction level and system routine functions required to support the AWACS processing functions.

Based on this input, the language group will develop a language and implement this language on the RADCAP testbed. The system software activity will implement routines to support the command language. The applications program will then be run on the testbed using, where possible, nonsynthesized data as input. The machine activity will be monitored to gather statistics on utilization, identify system bottlenecks, and determine the efficiency with which the algorithms provide solution.

The data collected will then be analyzed to determine where cost-effective improvements can be made to software and/or hardware in order to improve the cost-effective performance of the system. These changes will be incorporated into the system via micro-program or software routines. If the change is to be a hardware design, that design will be made to the gate level so that performance and cost-effectiveness determination can be made.

When the solution to the problem is finally refined, it will be contrasted with known sequential solutions.

Initially each of the AWACS data processing functions will be treated separately. The final task will then be to develop a system executive and integrate all the functions to reflect the real world.

REFERENCES

1. Batcher, K. E., *STARAN Parallel Processor System Hardware*, GER-15996, Akron, Ohio, Goodyear Aerospace Corporation, 19 November 1973.
2. Davis, E. W., *STARAN Parallel Processor System Software*, GER-15997, Akron, Ohio, Goodyear Aerospace Corporation, Akron, Ohio, 19 November 1973.
3. Rudolph, J. A., "A Production Implementation of an Associative Array Processor—STARAN," *1972 Fall Joint Computer Conference Proceedings*, December 1972, pp. 229–241.
4. Slade, A. E. and H. O. McMahon, "The Cryatron Catalog Memory System," *1957 Fall Joint Computer Conference Proceedings*, Vol. 10, pp. 115–120.
5. Fulmer, L. C. and W. C. Meilander, "A Modular Plated Wire Associative Processor," *Proceedings of IEEE Computer Group Conference*, June 1970.
6. Shore, J. E., "Second Thoughts on Parallel Processing," *NRL Report 7364*, December 30, 1971, page 7.
7. Kressler, R. R., C. E. Peet, Jr., and F. B. Frazee, "Development of an LSI Associative Processor," *Air Force Avionics Laboratory Technical Report 70-142*, August 1970.
8. Boyle, J. T. and C. A. Neugebauer, "Approaches to Custom Large Scale Integration," *Air Force Avionics Laboratory Technical Report 73-66*, March 1973.
9. Batcher, K. E., "Flexible Parallel Processing and STARAN," *1972 WESCON Technical Papers*, Session 1.

STARAN parallel processor system software

by EDWARD W. DAVIS

Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

This paper is concerned with the features and concepts of system software for a parallel associative array processor—STARAN.* Definitions of parallel processors have appeared often. Essentially they are machines with a large number of processing elements. They have the capability to operate on multiple data streams with a single instruction stream. STARAN is a line of parallel processors with a variable number of processing elements.

Along with the multiple processing elements, STARAN has a memory organization that allows access either by location or association. That is the address of a memory word can be used explicitly, or words can be selected by association based on their content. Processing elements can operate on data selected associatively, making the machine an associative processor.

An alignment, or permutation, network in the machine provides a flexible interconnection between processing elements. This network is used to align data in the memory with the processing elements requiring the data and to provide communication between processors. This results in an array organization, making the machine an array processor. STARAN is thus a true parallel, associative, array processor.

It is expected that one might be curious about the use of this machine: the operating system, language processing software, user program development, and execution control aids. This paper gives a brief description of software for all these purposes. Some parts will be recognizable as fundamental members of the software for other general purpose computing systems. Special development was required, however, to handle features unique to the parallel organization.

The programming language is new. It includes declarations for defining storage in the arrays and instructions for using the parallel and associative properties of the machine. Interactive execution control software has been written. It simplifies development and debugging of user programs. This software differs from conventional debugging tools by the extensions related to the array memory organization. Discussion of the language and control software, plus methods of interfacing STARAN to other machines, are the major points of the paper.

* TM, Goodyear Aerospace Corporation, Akron, Ohio.

STARAN SYSTEMS

STARAN is an operational computing system. The hardware architecture is described in a companion paper presented at this conference[1] and in other literature.[2,3] A particular installation and its potential use is described in a companion paper.[4] This paper is concerned with a description of the existing system software. There are two modes of operation. First, STARAN can be operated as a stand-alone parallel processing system. Peripherals for this mode typically include a card reader, line printer, paper tape reader and punch, and cartridge type disk unit. Second, STARAN and a cooperating, or host, machine can be operated in an integrated fashion. This means that: (1) commands to the STARAN disk operating system can originate in the other machine, (2) the storage system of the host is available to STARAN users for program or data storage, and (3) a single task can use both machines to satisfy its processing requirements. All peripherals belonging to a stand-alone STARAN and to the host are available when the machines are integrated.

This paper describes the software for the STARAN stand-alone mode of operation, then covers the additional software used with the integrated mode.

Since the STARAN processor architecture is detailed in a companion paper[1] only a basic diagram is given in Figure 1. The multi-dimensional access associative arrays and their controls are the main architectural features. The sequential control, a Digital Equipment Corporation (DEC) PDP-11 minicomputer, has a minor role in the architecture but is important for software considerations. Other architectural features are mentioned later in the paper.

SOFTWARE FOR STARAN STAND-ALONE MODE

Software for the STARAN stand-alone mode of operation can be discussed from the standpoints of the operating system, language processing, and execution control procedures.

Batch disk operation system

In this paper, an operating system means the collection of routines that give the user appropriate control of the com-

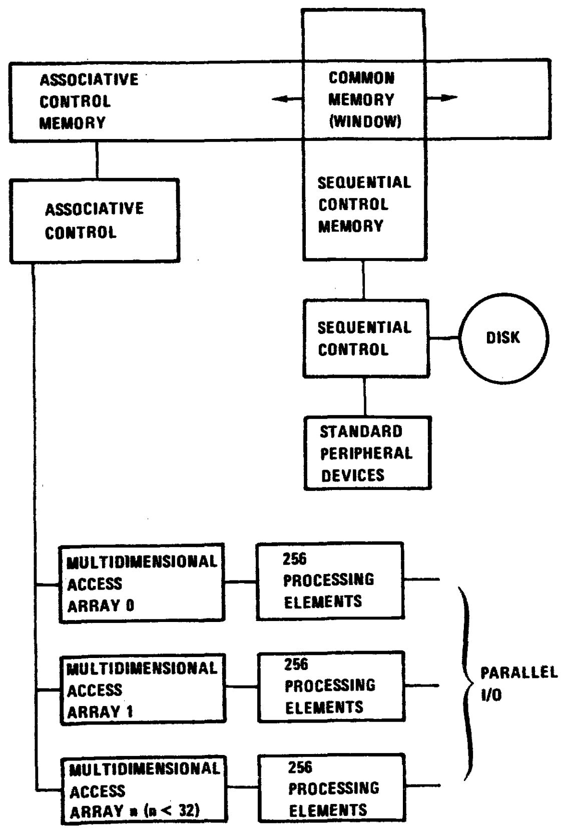

ASSOCIATIVE CONTROL MEMORY
COMMON MEMORY (WINDOW)
ASSOCIATIVE CONTROL
SEQUENTIAL CONTROL MEMORY
SEQUENTIAL CONTROL
DISK
STANDARD PERIPHERAL DEVICES
MULTIDIMENSIONAL ACCESS ARRAY 0
256 PROCESSING ELEMENTS
MULTIDIMENSIONAL ACCESS ARRAY 1
256 PROCESSING ELEMENTS
MULTIDIMENSIONAL ACCESS ARRAY n (n < 32)
256 PROCESSING ELEMENTS
PARALLEL I/O

Figure 1—STARAN block diagram

puting system, inform him of system status, provide input/output (I/O) facilities, and provide access to system programs. STARAN features a disk operating system (DOS) and has a batch processing capability. The batch command stream can be assigned to any character input device, allowing control to originate at the control console or from a user's file on the batch device.

The disk is a file structured bulk storage medium. All system software is resident on the device for easy, rapid access by the user.

Listed below are the standard programs supplied with the DEC PDP-11 batch system:

Program Name	Function
MACRO	Macro-assembler
LINK	Linker
LIBR	Librarian
PIP	File utility package
EDIT	Text editor
ODT	On-line debugging package
FORTRAN	Fortran compiler

These programs are not discussed further since primary emphasis in this paper is on the STARAN-related software that has been added to the above list to build the STARAN disk operating system.

One general rule used in software development was to avoid changes to the basic DEC batch system. This rule was intended to simplify any future change to a new DEC release.

Language processing

APPLE—Programs for STARAN are written in the APPLE* assembly language (**A**ssociative **P**rocessor **P**rogramming **L**anguag**E**).[5] This language has some mnemonics that generate one machine language instruction and others that generate a sequence of machine instructions. The one-to-many mnemonics generally implement a parallel algorithm for arithmetic or search operations using the arrays. Thus, APPLE is at a higher level than sequential machine assembly languages.

APPLE produces relocatable or absolute program sections and has a conditional assembly capability. Groups of instructions in the language are listed below:

1. Assembler directives
2. Branch instructions
3. Register load and store
4. Array instructions
   a. Loads
   b. Stores
   c. Associative searches
   d. Parallel moves
   e. Parallel arithmetic operations
5. Control and test instructions
6. Input/output (I/O) instructions

Most of these groups of instructions resemble those of other typical assemblers. The unique group—array instructions—deals with operations on the multi-dimensional access arrays and the registers in their processing elements (PE). Some general comments apply to all the array instructions listed above. Operations take place only on arrays enabled by the array select register.[2] Fields are of variable length within each array word and are defined for various instructions by field pointers and length counters. The common register, a part of associative control, can contain an operand, which is used in common by all selected array words.

More detail is presented below on the array instructions; i.e., loads, stores, associative searches, parallel moves, and parallel arithmetic operations.

The "load" array instructions load the processing element (PE) registers or the common register with data from arrays. Logical operations may be performed between the current PE register contents and the array data. The language has mnemonics for the common logical operations, while the machine supports all 16 functions of two logical variables.

* TM, Goodyear Aerospace Corporation, Akron, Ohio

A given load instruction can increment, decrement, or leave as is an array field pointer. Thus, a single one of these instructions can load registers, perform logic, and change pointer values. Operations to set, clear, or rotate the PE registers are included in this group.

The "store" array instructions are used to move PE or common register data into the arrays. A mask feature is provided that allows writing only in mask enabled array words. As with the load instructions, logical operations may be performed between the current PE registers contents and the array data. Also, the array field pointer can be incremented, or left unchanged.

The "associative search" array instructions allow the programmer to search for particular conditions in the arrays. Only those words enabled by the mask register take part in the searches. Searches can be performed that compare a value in the common register with a value in a field of all array words. Another variety of search compares one field of a word with a second field of the same word for all array words. Comparisons can be made for such conditions as equal, not equal, greater than, greater than or equal, etc. Maximum and minimum searches also can be performed. Combinations of searches yield such functions as between limits and next higher. Additional mnemonics in this group are provided to resolve multiple responders to the searches.

The "parallel move" instructions are provided to move an array memory field to another field within the same array word. As with searches, a word is active for this instruction only when enabled by the mask register. Types of moves are direct, complement the field, increment or decrement the field, and move the absolute value.

The "parallel arithmetic" array instructions allow the programmer to perform parallel operations in the arrays. These operations are subject to mask register word enabling. Arithmetic can use a value in the common register as one operand and a value in a field of all array words as the parallel operand. Alternatively, one field of a word can be arithmetically combined with a second field of the same word for all array words. Operations supplied by APPLE are add, subtract, multiply, divide, and square root.

Macro—A macro language is provided to increase the user's flexibility at assembly time.[6] The macro language has a large set of arithmetic, logical, relational, and string manipulation operators. Adding macro variable symbol handling, conditional expansion capability, and ability to nest macro calls make it possible to write powerful macro instructions. System and user macro libraries have been implemented.

Benefits to the user are the ability to define new mnemonics, redefine existing mnemonics, and conveniently generate standard instruction sequences.

Mnemonics have been added to the basic APPLE language by including macros in the system library. Primarily, the added mnemonics are floating point instructions. They are fixed field length operations in both single and double precision.

Building Load Modules—Software used to convert source language programs into executable load modules includes an APPLE assembler, macro-preprocessor, and relocating linker. Figure 2 shows this software and the flow of programs or modules through it.

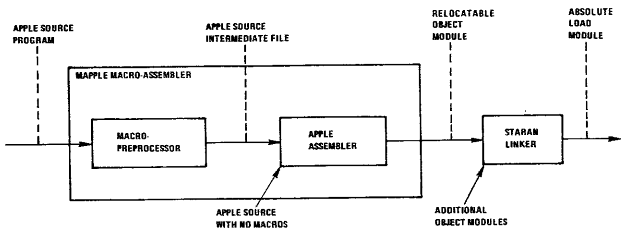


Figure 2—Language processing software

Building load modules begins with the original program written in APPLE. This source program may contain macro instructions. Translation of the source into a machine language object module is by MAPPLE (APPLE assembler with Macro-preprocessor on the front end). If it is known that the source program does not contain macro instructions, it is possible to input the source directly to the APPLE assembler.

A relocatable object module is converted to an absolute load module by the STARAN linker. Multiple object modules may be input to the linker since it has the function of resolving symbols defined across object module boundaries (global symbols) as well as adjusting addresses for relocation.

Use of the language processing software is fully described in the STARAN User's Guide.[7]

Execution control

Execution control software is discussed below, covering loading, executing, and debugging programs on STARAN. Four modules are involved: the loader, STARAN program supervisor, debug module, and control module.

Loader—Output of the STARAN linker is shown in Figure 2 as an absolute load module. The loader has the straightforward task of moving a load module into STARAN control memory beginning at the address specified in a text block. Options on loading are to load and not execute or to load and begin execution either at an address given with the load module or at one given with the load command. The load module is accessible from a user program to enable calling for a load from an executing program. This means that overlay modules can be brought in dynamically.

STARAN Program Supervisor (*SPS*)—The SPS is the software interface between the associative and sequential portions of STARAN. This module has services for system users when programming in APPLE and when programming a PDP-11 routine to interact with an APPLE program.

For the APPLE program, SPS makes the I/O instructions of the disk operating system (DOS) available, provides a program overlay capability, and provides a programmable interrupt to a PDP-11 routine. The PDP-11 routine inter-

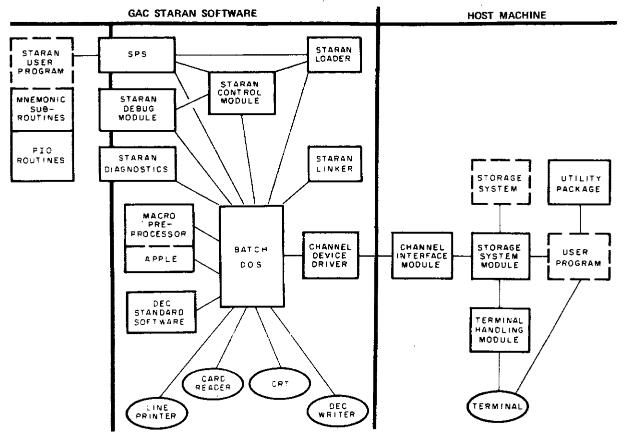


Figure 3—System software diagram

acts through a software link, which receives the APPLE interrupts, and through the issuing of control information to the associative control logic.

In addition, SPS supplies interface services. It transfers data between associative and sequential memory through the common memory window (Figure 1). SPS also fields associative processor error interrupts.

Concurrent execution of associative and sequential routines, with interaction, is made possible by SPS.

STARAN Debug Module (SDM)—The SDM helps the user debug APPLE programs by giving him control of the execution of the program being debugged, and access to memory and registers. Such features as single step, trace, and breakpoint provide good execution control. Dumps of all memory areas can be taken, with both word slice and bit slice available for the multi-dimensional access arrays. All memory locations also can be modified.

STARAN Control Module (SCM)—This final operational module is the interface between the user and execution of a STARAN program. By running SCM, the user enters a mode in which STARAN related commands are recognized. Such commands as start, halt, and continue execution are processed directly by SCM. When the load command is used, SCM passes control to the loader for that function. If debug aids are needed, a simple command adds all debug module features to SCM.

All the operational software modules are described more fully in the STARAN User's Guide.[7]

SOFTWARE FOR THE INTEGRATED MODE

General

The integrated use of the STARAN parallel processor and a host sequential computer makes additional software necessary. One major concern is the interface between the computers; this requires a software module in both machines. A second concern involves reasonable ease of use for the integrated mode; procedure packages are added as needed to satisfy this concern.

Figure 3 is a block diagram of the software modules in STARAN and a typical host machine. Interface software can be seen as the channel device driver in STARAN and the channel interface module in the host. Routines that might be added to simplify operation in the integrated mode are the storage system module to provide access to the host's storage, a terminal handling module to provide smooth interaction with a terminal user, and a set of utilities.

The STARAN/HIS-645 software

Figure 4 shows the relationship between software modules in STARAN and the HIS-645, which runs under the Multics time-shared operating system.[8] This facility exists at Rome Air Development Center (RADC), N. Y. and is described in a companion paper presented at this conference.[4] As indicated, Multics contains three categories of software: command level, user process, and system related. Command level software is brought into execution by user-supplied commands, as from a Multics terminal. User process software consists essentially of subroutines called from a user program. System-related software is the collection of routines that support use of the system, such as handling input and output, and are usually called indirectly by the user program.

Additional details on the design and use of this software are described in the STARAN/HIS-645 User's Guide.[9]

Interface Modules—The two modules for the interface, shown in Figure 4, are the 645 device driver in the STARAN

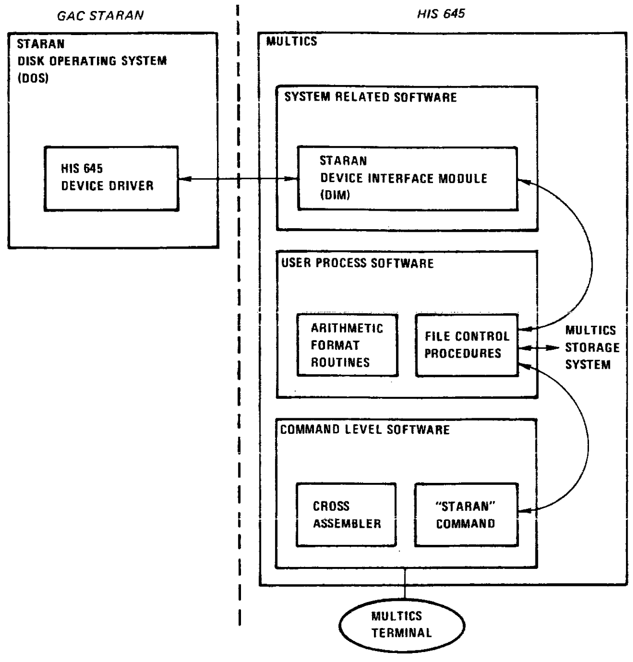


Figure 4—STARAN/645 system software relationship

batch disk operating system (DOS) and the STARAN device interface module (DIM).

The 645 device driver provides the interface between the DOS monitor and the 645 computer. It communicates with the monitor as do other device drivers for standard peripherals. If the device looks like an input for character information, then batch commands can come from it. The batch stream can be assigned to the device. This is the significance, for Multics, of the batch feature on the DOS.

In reality, the device treated by the 645 driver is used for much more than character input. The 645 appears as three logical devices.

One device looks like the disk, logically. The driver supports both ASCII and binary transfer modes, both formatted and unformatted. At any one time, up to 14 data-sets may be open on this device.

A second device looks like a card reader, logically. It is a read-only device with an ASCII transfer mode. This unit serves as the batch command stream input so a Multics user can control the system.

The third device looks like a paper tape punch, logically. It is a write-only device with ASCII and binary transfer modes. Job log output, in the integrated mode, is always assigned to this unit.

STARAN DIM—In Multics terminology, a device interface module (DIM) coordinates communications with a particular physical device. Data manipulation by the STARAN DIM assumes all Multics data is in character form. It converts characters into the form needed for output to STARAN and converts data received from STARAN into Multics character form. This means, for example, that Multics arithmetic data must be converted to a character form prior to output, and from characters following input. The conversion is done by a procedure superior to the DIM. The DIM also handles retransmission of bad data and reports a failure to its caller after a specific number of unsuccessful tries on the same data.

In the Multics software structure, the DIM is located in a position inferior to the file control procedures, shown in Figure 4 and described in the next part of this paper.

System Use Modules—The file control procedures (FCP) greatly simplify operation of STARAN from Multics. It enables a Multics user process (program) to interact with STARAN by initializing the interface, handling communication between the machines, and terminating the interface. The FCP also makes the necessary calls to the DIM to initialize and terminate the interface.

With FCP, a user process, executing in the 645, can call for STARAN, and it can pass commands, programs, and data to STARAN. The FCP raises the point at which the user becomes involved from sequences of calls to the DIM to a more symbolic call to FCP routines from the user process.

User involvement in the interface to STARAN is raised still higher from the user process to the Multics command level by a "STARAN" module. Essentially, this module is a supplied user process that passes parameters used in the terminal command to the FCP. The parameters identify the STARAN batch command stream input and output devices. The module calls appropriate FCP routines to establish interaction with STARAN.

In typical operation of STARAN from a terminal, this Multics command is used with STARAN commands also coming from the terminal. Initializing and terminating the interface are not a concern of the user. The Multics terminal becomes very similar to the STARAN control console when this module is used.

STARAN and the 645 differ in the lengths of their data representations. STARAN has a 32-bit control memory, while the 645 has a 36-bit word length. Arithmetic format routines are provided to convert either integer or floating point data between the 645 format and the format used by the DIM for transmission to STARAN.

A cross assembler has been written in PL/1. This is a functionally equivalent version of the MAPPLE assembler to be run in Multics. It is available to terminal users on the time-shared basis. It accepts APPLE and macro statements and produces STARAN object code in the Multics character format required by the DIM for transmission to STARAN.

STARAN/Σ5 integrated mode

A second method of interfacing STARAN with a host machine has been implemented in the Evaluation and Test Facility at Goodyear Aerospace Corporation. This facility has an XDS Σ5 as the host. The direct memory access capability of STARAN has been used to allow an 8K area of Σ5 memory to be used as STARAN control memory. Either programs or data may be stored here with control provided by interrupts between the machines. Software for this system is a communications library package with subroutines callable from FORTRAN or machine language in the Σ5.

CONCLUSION

A brief description has been given of software packages that compose the system for the operational STARAN parallel associative array processor. Also described is the additional software that makes STARAN operational when integrated with HIS-645 or XDS Σ5 sequential computers. The goal of all the software is to provide tools to use STARAN in the stand-alone and integrated modes. The tools are intended to increase convenience for the user and improve total system throughput.

Many modules have been discussed. Some of these are essentially transparent to the user, some may not be needed by certain users, and some may be required by all users. For stand-alone STARAN operation, the programmer must know APPLE and the use of the assembler and linker. He must be able to run the control module and load programs.

He will probably be interested in the debug module. The STARAN program superivsor is transparent for most users. It is not necessary to know any of the sequential control programs or languages.

REFERENCES

1. Batcher, K. E., *STARAN Parallel Processor System Hardware*, GER-15996, Goodyear Aerospace Corporation, 19 November 1973.
2. Rudolph, J. A., "A Production Implementation of an Associative Array Processor—STARAN," *1972 Fall Joint Computer Conference Proceedings* December 1972, pp. 229–241.
3. *STARAN Reference Manual*, GER-15636A, Goodyear Aerospace Corporation, September 1973.
4. Feldman, J. D. and L. C. Fulmer, *RADCAP: An Operational Parallel Processing Facility*, GER-15946B, Goodyear Aerospace Corporation, 21 December 1973.
5. *STARAN APPLE Programming Manual*, GER-15637A, Goodyear Aerospace Corporation, September 1973.
6. *STARAN MACRO Programming Manual*, GER-15643, Goodyear Aerospace Corporation, September 1973.
7. *STARAN User's Guide*, GER-15644, Goodyear Aerospace Corporation, September 1973.
8. Organick, E. I., *The Multics System*, MIT Press, 1972.
9. *STARAN/HIS-645 User's Guide*, GER-15641, Goodyear Aerospace Corporation, September 1973.

Some thoughts on associative processing languages

by WILLIAM W. PATTERSON

Rome Air Development Center
Griffiss AFB, New York

INTRODUCTION

Much effort has been expended in developing array and associative processors (AP's). The most notable of the former are Burroughs' ILLIAC IV and Honeywell's PEPE, while the present representative of the latter technology is the STARAN built by Goodyear Aerospace Corp. However, very little has been published on higher order languages which take advantage of the unique characteristics of these architectures. There is at least one effort to develop techniques which will extract the parallelism in ordinary FORTRAN code,[1] as well as a number of efforts to formally describe the parallelism in algorithms. Examples are in References 2 and 3. It is true that many algorithms can be put into efficient parallel code using these techniques; however, there is a large body of problems which must be re-examined and recast into new algorithms which match the parallelism of the machine to the natural parallelism of the problem. These new algorithms will require a new language which gives the programmer the flexibility to use the features of the machine directly. The PFOR language[4] developed for PEPE is probably the only existing language for an array processor, and some preliminary work for the RADC AP project[5] is the only published attempt on AP languages. This paper will look at associative processing from the point of view of a programmer who has tried to write programs for an AP, and therefore will propose constructs which are convenient for the programmer and not necessarily for the compiler writer. They do, however stem from a reasonable knowledge of the basic architecture of the AP, and hence will tend to parallel it.

It is necessary at this point to talk about the general architecture of the AP and, in the process, define some terms that will be used in the paper. The two main units in the AP are the control unit and the associative processing elements. The control unit includes a mainframe memory, which holds the programs, constants and common single valued variables; local arithmetic capability that can be used to perform processes which concern only the common variables; and control logic to drive the associative processing elements. The associative processing elements each consist of an associative word of 256 or more bits, and logic to process the data in that word. The associative word is divided into several fields of either fixed or varying length. Normally, if the fields are of varying length, they are defined by the variables of the problem. Since the conventional computer performs operations sequentially, the term "sequential" will be used in the sequel to distinguish the conventional computer; however, the modifiers "sequential" and "associative" may be left out if the context makes it clear which is meant.

LANGUAGE

In designing a language such as this, two choices are possible; one can design a complete new language, or he can modify an existing language to include the proper constructs. The latter approach was chosen because very few problems are completely associative and it is anticipated that most installations that include an AP will also have a conventional computer with facilities for communication between the two. This allows the programmer to write all his code in the same basic language, identifying the associative parts. The language chosen is PL/1 for several reasons, not the least of which is the author's familiarity with the language. However there are more cogent reasons; the first is that the basic block structure of PL/1 lends itself to segregation of sequential and associative tasks into separate routines. Other reasons include its basic self-documenting qualities, its extreme flexibility which allows its use for a large number of problems and the fact that it has a degree of parallelism already built in which might be exploited.

Declaration

The associative tasks in the problem should be segregated from the sequential tasks on a procedure level. This is easily facilitated by defining an associative procedure as qualified procedure much like the presently defined recursive procedure. The form of the procedure declaration statement would be:

```
label: PROCEDURE (parameters) attributes
   ASSOCIATIVE nr_entries;
```

or alternately:

```
label: PROC (parameters) attributes ASSOC nr_entries.
```

This statement would tell the compiler that procedure (label) should be compiled into AP code. The parameters and attributes fields are optional and follow the same rules as in PL/1. The optional nr_entries field tells the number of associative entries needed by this procedure,* where an associative entry is one of a number of identical sets of variables, each of which has a unique set of values and each of which will be processed in parallel with all the others. Associative entries are distinct from associative words, since it is quite possible to have more than one entry per word or to have one entry fill more than one word. However, the task of controlling these configurations is best left to the computer. This allows the programmer to specify an associative entry of the length appropriate to the problem, and conceptually think of an entry as an associative word (this being the case, no distinction will be made between the two in the sequel). An example of a set of associative entries is a radar track file which keeps a record of all the tracks being monitored by a radar set or system (for example, an air traffic control radar). Each entry stores all the information about one track, such as position coordinates, track quality and any keys or flags which give additional information about the vehicle being tracked.

It should be noted here that if a sequential procedure calls an associative procedure or vice-versa, and there are no data dependencies, then the two procedures can run concurrently on the two machines. Therefore, suitable WAIT statements must be inserted when the calling procedure needs data from the called procedure. This will cause the calling procedure to wait at this point in its execution until the procedure it called has finished, thus assuring that the data required is properly updated.

The question now arises as to what variables should be passed between associative and sequential procedures. The normal PL/1 convention is that a variable declared in a procedure is available to all procedures it calls but not to procedures that call it. This seems impractical in this case since any call from one type of procedure to the other implies that the data must be passed over a physical channel. For this reason, the author favors the restriction that only formally declared parameters be passed between different types of procedures, with the normal rules applying to calls between two procedures of the same type. This dichotomy should not be troublesome to the programmer, since he must know that he is working on two different machines.

In associative processing, there are two basic types of variables. The first type is the common variables and constants which are single valued and therefore are held in the mainframe memory. The second type is the associative variables which have a value for each of the associative entries. The distinction between these two types of variables will be made in the variable declaration of the associative procedure. The variables of the first type would be declared in the normal manner, where the STATIC attribute would indicate that the variable would be assigned a static location in the AP mainframe memory. Associative variables would be declared in the same manner except that the keyword ASSOCIATIVE (abbreviated ASSOC) would be appended after the declaration. This declaration then defines a field in each associative word. For example, the variable which represents the range coordinate of the tracks in the track file mentioned above would be declared:

```
RANGE FIXED BIN(x1,x2) ASSOCIATIVE;
```

This indicates an associative variable called RANGE, a fixed binary quantity of precision x1 (i.e., a field of x1 bits in each associative word) with fractional part x2.

Dimensionality of associative variables has to be handled differently than with sequential variables, of course. The first dimension of all the associative variables is the number of associative entries. Since this number is contained in the procedure declaration, it would not appear in the variable declaration. If arrays are desired in each entry, then, of course, the normal convention would apply. Clearly, if an array is passed from a sequential procedure to an associative procedure and is stored as an associative variable, then the associative declaration should not include the first dimension of the array.

Association

The major unique feature of the AP is the ability to make associations; that is, it is possible to make parallel comparisons either between a comparand held in the control portion of the AP and a field in each of the associative words or between two fields in each of the associative words. Normally the processing which follows is performed only on those words that give an affirmative response or, in the jargon "respond," or on a subset of these words. It is convenient to talk about the words that have responded as being active. There are two basic association actions to be performed. The first is to associate only on those words that are active, and thereby further reduce the number of active words. This will be performed by an ASSOCIATE statement. The second type of association is used to reactivate a number of words after a series of ASSOCIATEs. This is accomplished by performing an association on all associative words and is invoked by the ACTIVATE statement. Most AP's have the capability to associate on the basis of the three basic relational operators, greater than (>), less than (<), and equal (=).

An ASSOCIATE statement would consist of the keyword ASSOCIATE followed by a conditional statement which include conditions using one or a combination of the above relations or logical combinations of the conditions using logical AND (&), logical OR (+), and logical NOT (¬). As an example of this type of statement, suppose that the associative variables RANGE_DIST and AZ_DIST hold the polar vector distance of each track from a current radar return, and we wish to determine which of these tracks is within a certain distance from the point of the return, where

* If this field is absent, then the default would be that nr_entries = the number of associative words in the machine.

that distance is held in mainframe memory in locations called RANGE_WINDOW and AZ_WINDOW. The statement would be:

```
ASSOCIATE RANGE_DIST<RANGE_WINDOW
  & AZ_DIST<AZ_WINDOW;
```

This statement would leave active all associative words which contained a track which fell within the window and deactivate all others.

There are two special capabilities of the AP that should be included in this section. These are the capability to find the minimum value in a given field and to activate all words which have that value in that field and an analogous capability to find the maximum. These capabilities can be invoked by an ASSOCIATE statement as follows:

To find the minimum value:

```
ASSOCIATE MINIMUM variable_name;
```

and to find the maximum value:

```
ASSOCIATE MAXIMUM variable_name;
```

Where "variable_name" is the name of the variable assigned in that field.

The second type of association statement is used to reactivate a larger set of associative words. It is the ACTIVATE statement and it works the same as the ASSOCIATE statement except that the keyword at the beginning of the statement is ACTIVATE, and it activates all words that meet the condition, not just the previously active ones. In addition, the keyword without any condition has meaning and that is that all associative words should be activated. Pursuing the radar tracking example to illustrate the conditional ACTIVATE statement, suppose that you wish, after a series of associations which identified a small number of tracks, to reactivate all associative words that contain valid tracks and that each track entry contains a one bit flag called BUSY which is 1 for every valid track and 0 for all others. The statement:

```
ACTIVATE BUSY;
```

would activate all valid tracks.

There are instances when it is necessary to activate one and only one associative word in a group but it is not critical which word is activated. In this case, the associative processor has the provision to activate the first word which meets the conditions of the search. This provision can be invoked by adding the keyword FIRST to the ASSOCIATE or ACTIVATE statement. For example, suppose that in the radar tracking problem, we wish to establish a new track, using the first empty word. The word can be activated with the statement:

```
ACTIVATE FIRST ¬BUSY;
```

Another feature that is quite useful is the capability to select a subset of the active words for execution of a short series of instructions without deactivating the other words. This can be invoked by using a FOR statement, which has the same syntax as the normal IF statement, except that the keyword FOR is substituted for IF. This signifies that all associative processing elements which meet the condition would execute the code between the THEN and END brackets. If an ELSE portion is included in the statement, it would be executed by all active words which do not meet the condition. This means that both parts of the statement are executed each time the statement is encountered.

Assignment

The last type of statement that will be discussed is the assignment statement. The simplest type, of course, is the move:

```
X=Y;
```

Let us now consider the four possible combinations of variables. If both X and Y are common variables, then the move is carried out in the normal manner. If both X and Y are associative variables then the statement constitutes a move of data from field Y to field X in each of the active words. If X is an associative variable and Y is a common variable, the statement constitutes a broadcast of the data in location Y to field X in all active associative words. If X is a common variable and Y is an associative variable, only one word can supply data, since there is only one location in mainframe memory to receive it. The source chosen is field Y of the first active word.

For more complex assignment statements, containing two or more variables on the right side of the equal sign, it is clear that if any one of the variables is associative, then the operation must be performed in the associative processing elements. Once the answer is found, the assignment will follow the rules shown in the preceding paragraph. For example, suppose we wish to calculate the distances used earlier for association, given that the information on a new report is stored in a structure in mainframe memory called REPORT. The code would be:

```
RANGE_DIST=ABS( REPORT.RANGE-RANGE );
     AZ_DIST=ABS( REPORT.AZ-AZ );
```

Since RANGE and AZ are both associative variables, these calculations would both be performed in the associative processing elements; and since RANGE_DIST and AZ_DIST are both associative variables, the result would be transferred directly to the proper fields in each associative word.

CONCLUSION

This paper has proposed language forms, based on the PL/1 language, which will give the programmer the capability to directly use all of the features of an associative processor without having to revert to assembly language coding. Many of the statements will translate into one or two lines of machine code, but this is necessary to use of the full power of the machine. The capability is important, because there are a class of problems which will require new algorithms to effectively use the machine by matching the parallelism of the machine to the parallelism of the problem.

REFERENCES

1. Lamport, L., "The Coordinate Method for the Parallel Execution of DO Loops," *Proceedings of the 1973 Sagamore Computer Conference*, pp. 1–12, August 1973.
2. Muroaka, Y., *Parallelism Exposure and Exploitation in Programs*, Ph.D. Dissertation, U. of Illinois, Urbana, 1971.
3. Ramamoorthy, C. V., J. H. Park and H. F. Lee, "Compilation Techniques for Recognition of Parallel Processable Tasks in Arithmetic Expressions," *IEEE Transactions on Computers*, Vol. C-22, pp. 986-998, November 1973.
4. Dingledine, J. R., H. G. Martin and W. M. Patterson, "Support and Operating System Software for PEPE," *Proceedings of the 1973 Sagamore Computer Conference*, pp. 170-178, August 1973.
5. DeFiore, C. R., A. A. Vito and L. Bauer, "Toward the Development of a Higher Order Language for RADCAP," *Proceedings of the 1972 Sagamore Computer Conference*, pp. 99-112. August 1972.

User/system interface within the context of an integrated corporate data base

by GENE ALTSHULER

Stanford Research Institute
Menlo Park, California

and

BERNARD PLAGMAN

The Federal Reserve Bank of New York
New York, New York

Man will constitute to a higher and higher degree the limiting factor in man-computer achievements.[1]

INTRODUCTION

The world to come may be viewed as demanding the making of more decisions per unit time, the taking into account of more variables per decision, and the greater commitment of resources per decision. . . . At the same time that the environment is pressing for faster decisions, new esoteric technologies are demanding longer planning cycles, earlier and higher commitments of resources, and better integration and coordination of a greater number of interacting elements that make up new systems.[2]

The recognition of the truth in this statement is not by any means recent. Nevertheless, only since the application of the digital computer to the commercial environment has the potential of a solution existed. Not until the invention, application, and maturity of this technology was it deemed possible that methods could be found to deal effectively with the problem. It was realized that the unaided human being could not hope to deal with the complex emerging environment of modern business. Numerous works have dealt with the limitations of human information processing (for example, References 3, 4, and 5).

It was only some five years after the application of the computer to commercial data processing that the notion evolved that the ultimate extension of computerization was not simple and repetitive transaction processing. It was realized that managers process information as do clerks—though certainly not in transaction form, and against a wider universe—and that computer systems can and should be developed to aid them in this process. The term Management Information Systems arose to encompass this concept, and much was written and many attempts were made to develop MISs. More often than not these attempts failed. Early proponents of the MIS concept did not realize that the task of creating, maintaining, and accessing extremely large and structurally complex data bases to be shared by multiple users with diverse information requirements would necessitate all of the following:

- Extremely sophisticated resource and task management software.
- Very fast hardware logic and peripherals.
- Large-scale, inexpensive memory.
- Software to create, maintain, and access vast amounts of data.
- Supportive subsystems to integrate and augment the human user effectively.

Today, a search for even one example of the prophesied total or integrated management information system might prove futile. The unavailability of software to create, maintain, and access corporatewide data is one fundamental technical reason for the failures. Recognition of this state of affairs has led to great activity toward the development of constructs,[6,7,8,9] requirements,[10,11] and specifications[12] for what have been termed Data Base Management Systems (DBMS). Numerous DBMSs are in various stages of implementation, development, and use.[13,14,15] It is important, however, to view the DBMS as a necessary but not fully sufficient component for the creation of information systems that will provide support for human problem solving. The task of problem solving is essentially a human one, and we do not in any sense forecast that it will be totally assumed by computers. What is recognized is that the human problem solver, unaided, is not able to cope sufficiently with the data that are available to be brought to bear on today's problems. On the other hand, the machine alone cannot identify the problems so as to begin to solve them. What we propose is a man-machine symbiosis where a synergistic relationship is established. When the problem solver or decision maker has been formally and implicitly introduced into the system definition, an interface can be clearly established between man and machine. It is to this interface, embodied in the concept of the User System Interface (USI), that the substance of this paper is directed. Within this context the aim of this paper is to:

- Introduce the concept of the USI via the decision process that it must support.
- Architecturally place the USI concept as an element within the ICDB.
- Discuss some design considerations of the USI.

THE DECISION PROCESS

Many researchers have published papers relating to the construction of conceptual frameworks of human problem-solving activities, for example Simon,[16] Newell et al.,[17] Gerrity,[18] and Miller.[19,20,21,22] For convenience we refer to their models as describing the "decision process."

Barkin and Lasky[23] have synthesized much of the preceding work and have established a sequential decision process that succeeds in subdividing the whole process into three distinct phases:

1. Intelligence Phase	Identification of the problem
2. Design Phase	Generation of alternative solutions
3. Choice Phase	Evaluation of alternatives and selection of the final solution.

Miller's work is closely related to this structure, for it can be viewed as subdividing the above three broad phases into specific tasks:

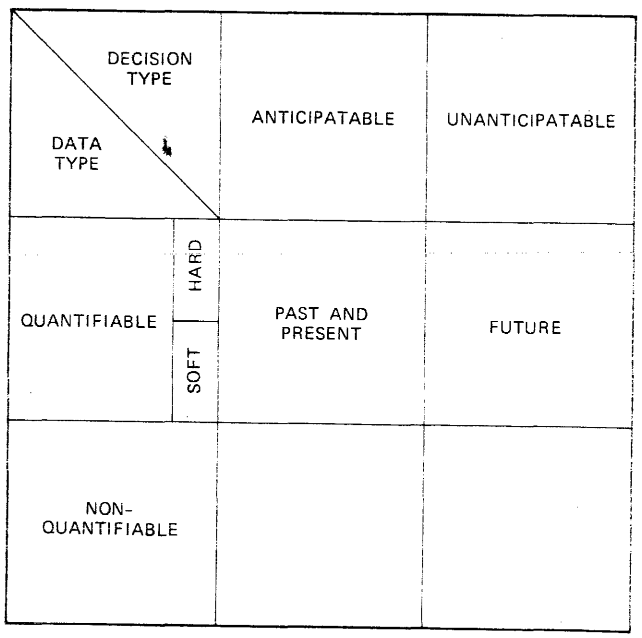
1. *Intelligence Phase*
   a. *Status Inquiry*—Equivalent to simple-inquiry. The operation involves retrieval or update by a unique identifier and/or attribute of interest.
   b. *Briefing*—Request for information about what is being accomplished or what has been accomplished, usually according to a set of categories about subject matter of interest or responsibility of the specific user.
   c. *Exception Detection*—Comparison of briefing information (what is) with planning information (what should be) and interpretation of the deviations. An exception exists if the user decides to take action, even if that action is further inquiry.
   d. *Diagnosis*—Posing of test questions to be answered, leading by a logical process of exclusion to the source of the gross symptoms.

2. *Design Phase*
   a. *Construction*—The building of new systems based on selected alternatives. The supporting system does two main chores: It applies the rules and constraints to each design action taken, and it remembers the work already done by the user.
   b. *Evaluation/Optimization*—Possible graphical simulation of systems, where the problem solver uses a graphic lanaguage to construct a model that will simulate the problem set.

3. *Choice Phase*
   a. *Planning/Choosing*—The matching of requirements sets (what do I have to do?) against resource sets (what do I have to do it with?). This is, in effect, multiple-category statistics matching.
   b. *Discovery*—Such operations as selective browsing through a trail of references in search for ideas that lead to concepts of trade-off structure and complex system behavior, or examination of different slices and cuts of quantitative data from a variety of experiments to derive a "hunch" as to control variables that might explain diverse phenomena. The key is that the user may not know in advance just what he is going to ask for or even exactly how he will make use of what he is exposed to.

The decision process, as outlined in this sequential list of activities, is the process that the USI must support, for it is inherent in almost all activities humans engage in. Individual modules of the USI will provide specific support to some or all of the activities in the decision process. Languages, for example, will provide support across the entire process, while simulation tools can assist in the design and choice phases.

Present-day computers (systems) are primarily designed to solve preformulated problems or to process data according to predetermined procedures.[24] How then do we deal with problems in which the time-frame for solution or the unique-



DECISION TYPE / DATA TYPE		ANTICIPATABLE	UNANTICIPATABLE
QUANTIFIABLE	HARD	PAST AND PRESENT	FUTURE
	SOFT		
NON-QUANTIFIABLE			

Figure 1—Decision model

ness of situation does not allow, or even call for, the specification, coding, testing, debugging, and documentation of procedural logic?

Decision can be classified as falling into two types: Anticipatable or preformulatable decisions are those that we know in advance must be made in the future, and unanticipatable decisions are those that cannot be predicted or foreseen. Although we cannot identify all the specific problems that fall into the two categories, we can nevertheless establish some general characteristics. The identification of these two types of decisions and the need to integrate data into the decision process are expressed in the model in Figure 1. Anticipatable decisions tend to be repetitive, thus susceptible to preestablished logic and less significant in terms of resources allocated. The converse is true for unanticipatable decisions.

Data relate directly to decisions; neither decision logic nor data alone are sufficient to support a decision. Data are not homogeneous; for our purposes they are divisible into quantitative data, which can be expressed numerically, and nonquantitative data, which cannot be expressed numerically. Quantitative data may be thought of as being hard or soft, soft being correlative or having a low level of confidence, such as Gross National Product estimates or the Consumer Price Index, and hard having a very high level of confidence, such as the units of production in a manufacturing facility.

Nonquantitative data are things like the mood of the corporate stockholders, employee morale, or the legal constraints on a corporation for acquisition. Such data are certainly essential as input to the decision process and may be stored and retrieved by a computer, against key words, in narrative form, but they are not susceptible to algorithmic manipulation. Because of the lack of ability to deal precisely with nonquantitative data, the best we can hope for it is to bring more and more of such data into the realm of the quantifiable, so as to increase the scope of our decision processes.

ARCHITECTURAL PLACEMENT WITHIN AN INTEGRATED CORPORATE DATA BASE

The ICDB is not a system. It is a concept under which systems should be implemented. The ICDB concept and the USI element have been evolved to handle the duality of problems in supporting anticipatable and unanticipatable decisions with quantifiable and nonquantifiable data, in part because proponents of DBMS software have been attempting to establish this subelement as a panacea. As we pointed out earlier, considerable work and attention have been paid to the DBMS following recognition of the fact that the complexity of creating, maintaining, and accessing a repository of corporatewide data was an order of magnitude higher than available software could handle.

The ICDB is formally defined as:

> The consideration of the collection, storage, and dissemination of data as a logical, centrally controlled, and standardized utility function.[8]

Any attempted implementation of the ICDB concept implies the development of five subsystems or elements:

1. The Data Bank—The logically centralized repository of all the data used by a corporation.
2. The Data Dictionary/Directory System—The repository of all the definitive information (meta-data) about the Data Bank, such as characteristics, relationships, and authorities.
3. Data Base Administration—A machine-aided human function, with responsibility and authority over all data-related activities.
4. The Data Base Management System—A software function performing the storage, retrieval, and maintenance of data.
5. The User/System Interface—The necessary subsystems to permit multiple classes and types of users to direct the system to effectively structure and associate available data into information and thus to communicate with and fully utilize the system's resources.

At the beginning of this paper, we stated that "today, a a search for even one example of the prophesied total or integrated management information system would prove futile." We stand by this statement even though one can point to certain very powerful data management software implementations, existing dictionary/directory systems, a variety of generalized query languages, and a fair number of data base administrators in place.

What has been lacking is that the conceptual construct to integrate all these elements has not been established and there has been a paucity of attention and work in the area of user/system interface. (We will concede the point that the need for the USI has been identified for many years, numerous times.)

Architecturally, aspects of the USI affect all five ICDB subsystems or elements, since ultimately all aspects of the structure must service the user. For instance, the DBMS must be able to retrieve data that may physically reside in only one form and remap them, as specified, into many different forms. Further, the DBA must provide the data in the first place (data collection) and tune the system so that all users are served optimally.[25] But the actual execution of the functions embodied in these two subsystems is essentially transparent to the end users. Therefore, the remainder of this paper focuses on the interface points of the various subsystems that are not transparent to the end users. Two of these are the user languages and the decision process augmentation tools.

USER LANGUAGES

In all probability, because language is so much a part of our everyday lives, we tend to view it as a homogeneous entity. This assumption is invalid, for in reality a language is employed by various classes and types of users, each user applying it to different depths and in different ways. For example, an engineer and a psychologist may both use

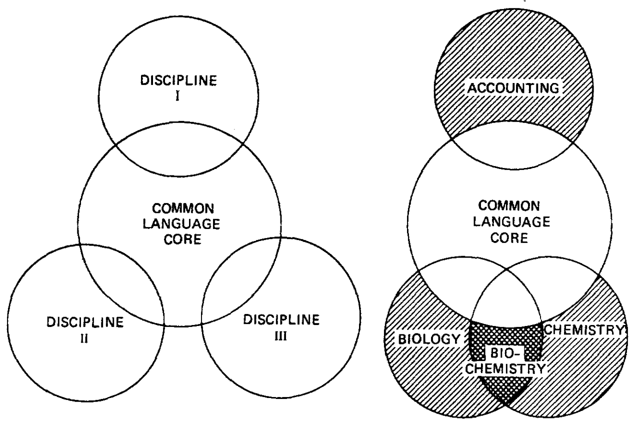


Figure 2—Natural language subset interaction

English as their natural or native language, and in their dealings outside their respective fields virtually no communication problems should arise (assuming both possess natural language competency). But extreme problems will arise when either attempts to cross over into the other's discipline (assuming no more than average familiarity with the other field). This is so because each discipline tends to create, hopefully out of necessity, specific jargon or nomenclature unto itself. The practitioners of these disciplines, then, tend to communicate in subsets of their natural or native languages. This characterizes language users into types centering around their expertise, the types of users being as varied as the specific professions, such as accounting and engineering.[26]

The Venn diagram in Figure 2 portrays graphically a simplistic situation. The "common core" is the native language base that is essentially used by all those to whom the language is common, no matter what their specific disciplines or even in the absence of a specific discipline. Further, disciplines may have varying degrees of mutual dependency, among themselves and with the common core.

At the extremes we may find mutually independent disciplines with virtually no overlap with the common core. A major problem can arise if an enterprise encompasses a multiplicity of mutually independent disciplines with minimal overlap with the common core, for then the vocabulary support essential in the user language area will tend to be significant. A second major problem may arise when discipline crossover takes place, in that different nomenclature may be associated with the same entity. A solution to the problem of a multiplicity of user types using the same data bank is a synonym resolution capability, in effect a complex-to-simple mapping.

A second dimension to the problem is that we have varying levels of application of a specific profession or discipline. For example, a bookkeeper, a cost accountant, and a company controller, or a nurse, a general practitioner, and a specialist. Recognition of such hierarchies characterizes language users, even within a type, into broad classes. Extrapolating on the work of Senko[27] we have structured users into three classes, for the purpose of directing our research into user language development:

1. Structurally independent
2. Structurally parametric
3. Structurally dependent.

The structurally independent user is generally characterized by his lack of interest in the storage structure of data and his interest in the specific data values and sometimes specific attributes of items of metadata (data about data, e.g., frequency of update, source). This class of user is usually personified by a manager who would, in all probability, have an information model in mind. The work into natural language recognition and programming is directed at this class, in that the ultimate objective is the ability to express a request for information in natural language and have the system do all the work necessary to interpret the request, search the data bank, select the necessary data, structure and/or associate the data, and return the information.[28]

The structurally parametric user is characterized by a limitation on what he may want to see data about (a finite subset of the data bank) and the number of ways he will want to see this subset structured or associated. The requirement is for a specific logical view, not a physical one. Thus the parametric user will typically invoke one of an array of preestablished transactions varying only in the values supplied at the time of retrieval. For example, someone within the purchasing department might request the part numbers of all the parts produced in plant three that are used in assembly A and have an inventory lower than four weeks production. Such a request can generally be anticipated in advance; for example, a boolean query on "X" of "N" fields in an employee record, so a screen display or hardcopy report can be prestructured and invoked by a transaction macro with associated parameters. The parametric user can reside anywhere within an organization.

The structurally dependent user is one whose basic concern is the physical or storage structure of data and their characteristics and attributes and who has little or no concern with specific data values. His primary objective is to optimize repetitive procedural manipulation of data and not to interpret or use the results. Jobs that typify this class are the programmer and analyst where a data base administration function does not exist or, of course, the data base administration function, if it is in place.

It is important to note that an individual within a job may slide between specific classes or users, on the basis of the function of the job he is performing at the moment. For example, an analyst when looking into the feasibility of developing a system will, on the basis of user requirements and specifications, be operating as a structurally independent user. At this time his prime concern will be to see whether the proper data elements exist within the data bank to satisfy his clients' needs. On the surface it would seem that this task could be simply supported with an absolute minimum of data retrieval, but in reality multiple queries embodying substantial amounts of data might be required. For example, if an analyst in a manufacturing environment were developing a system for the credit department with the intent of allowing the order clerk to approve a shipment, two fields would have to be accessed: customer balance and credit limit. But in reality there might be multiple fields labeled customer balance, each potentially representing different data values based on different frequencies of update and calculation algorithms. For example, the files might be so structured that customer balances were maintained for each product line, and a separate field might have been established by the marketing department to represent the average monthly customer balance. In all probability all these fields would be labeled simply "customer balance" (to each of these applications, one of these fields would be the customer balance required). Thus a number of attributes about a field would have to be called for by the analyst before he could make the determination whether to use an existing field or to create a new field. (Depending on organizational structure and the establishment of a data administrator function, this type of determination might not be performed by an analyst.) Once into the general design of a system, the analyst would have to operate as a structurally dependent user, for to instruct the ICDB as to the proper data structures (logical representations) he would have to be cognizant of the storage structures (physical representations) embodying the elements with which he was concerned and the exact physical characteristics and attributes of these data items.

Whether it would require many languages or one to satisfy the above requirements is not of immediate concern. What is important is that we provide the facilities for multiple types of users to operate on the same data without attempting to force these users out of their own natural language subsets. Further, it is important that the language facilities provided support multiple classes of users, so that query, parametric, and meta-data requests can be satisfied.

USER SYSTEM INTERFACE AUGMENTATION TOOLS

A reasonable amount of work has taken place in the area of operands, possibly because they are more tangible and thus more susceptible to definition. We have identified the need for three types of operands: arithmetic, graphical, and modeling.

Arithmetic operands are by far the most simplistic; they relate closely to the original rationale behind the development of the digital computer, that is, to the ability to perform basic arithmetic operations with a machine, orders of magnitude faster than a human being could unaided. Basic arithmetic operations are continuously performed on data under the preestablished logic approach. But when we are dealing with unanticipatable decisions, the time frame cannot accommodate the development of this logic, and thus we must be able to either establish it on an ad hoc basis or presupply it, and possibly both.

The concept of the ad hoc establishment of procedural logic is infinitely more difficult than that of attempting to anticipate needs and presupplying the needed tools; therefore we feel that we should approach the problem via the anticipation of generalized needs and the prior establishment of a catalog of tools. (The discipline that attempts to deal with the ad hoc establishment of procedural logic is natural language programming; the best estimate for substantive accomplishments in this field is from ten to twenty years. On the other hand, the selection of bits and pieces of preestablished procedural logic from a catalog and the structuring of these into a program may be considered the first step in this development, although the ultimate goal is for this process to take place automatically on the basis of a problem statement expressed in natural language.)

Examples of arithmetic operands to be supplied range from the most basic, such as addition, subtraction, division, and multiplication, to some of the more sophisticated, such as confidence limits and standard deviations. All of them would be callable via macros and capable of accepting as input the subset of data the user has selected and retrieved from the data bank.

The need for graphical operands is supportable, on the basis of observation of the way people extract and synthesize information from masses of data. In fact, it has become obvious to observers that people have a mechanistic inability to extract the essence or primary aspects of large tables of data without resorting to graphical or other means. Thus people pictorially represent large collections of numeric data, and upon the completion of the "picture," relationships, trends, and key factors become almost obvious. Computers perform graphical operations almost embarrassingly well, and this is one of those functions better left to the machine than to the human being. What must be supplied is, again, a catalog of graphs—curves, histograms, pie charts, or scatter diagrams—plus the ability to contract or expand the axes and to store, retrieve, overlay, and combine such graphs.

Lastly, and placed here because we are furthest away from its realization on the basis of the state of the art, is interactive modeling and simulation. Again, observations of people involved in the process of decision making have given rise to the understanding that, when one is faced with a problem involving a number of variables, intuitive modeling or simulation—or both—takes place. For instance, when we are faced with the relatively simple task of purchasing a number of different items available only at several different locations, a simplistic but identifiable transportation model is constructed by taking into account current location, available conveyance, and such constraints as one-way streets and closing times of various purveyors, and a hopefully optimal route is intuitively plotted.

Interactive computer-aided modeling is a necessary adjunct to the use of arithmetic and graphical operands, for it would allow the users to play "what if" games with extracted data, identify and isolate key relationships between variables, and identify and discard meaningless or insignificant variables. Further, by the imposition of the formalized discipline of modeling, it would aid in the identification of all necessary variables. The key difference between the modeling/simulation operand being discussed and the modeling and simulation tools currently available is that the former should be interactive and be accessible via the available query languages. Thus we would be able to use these tools without resorting to specialized languages and be able to operate in a conversational mode.

CONCLUSION AND SUMMARY

The USI is not a new concept. It is the collection of multiple existing concepts into a new environment, the ICDB. By establishing an ICDB environment and providing appropriate interface mechanisms, the various phases and steps of the decision process can be supported. It will not suffice to develop the DBMS and feel that we have "solved the problem." It is necessary to recognize that the end user must be served by facilitating the proper support of the decision process with the necessary data. This support will come in the form of user languages and specific types of augumentation tools collectively referred to as operands.

The essential points to be realized are:

1. The concept comprises five elements, which, when viewed collectively, form a unified whole. Certainly, each element, in and of itself, can be used effectively by an organization and to varying degrees can alleviate some of the problems now faced. There are two basic reasons for viewing, planning for, and implementing the concept as a whole:
   a. The concept is synergistic, in that the whole is greater than the sum of its parts. A data bank is worth more to an organization when it can be fully utilized by all those who require access to the data stored within it. Further, extremely flexible user languages will truly begin to return their cost of acquisition when they have a full-fledged data bank to access and a DBMS to structure and retrieve the data.
   b. EDP is evolving toward large data bases and multiple modes of reference. The primary question is that of the cost/benefit trade-offs and the proper growth rate at which to approach the objective. It is almost unquestioned that the approach must be via a disciplined, coordinated concept.
2. It is the decision process that we are trying to aid and support and not a specific level within the hierarchy of the organization, a specific function, or that nebulous and elusive figure called the manager. The decision process is the common denominator among all users. If this process is facilitated, those who carry out the basic functions of a firm will be aided and supported, and there will be no need to identify where within the structure they reside.
3. The ICDB concept is so broad that we could not begin to cover it in a single paper. This is the third paper in a series. We had originally planned to write one paper on each of the five basic elements, but while the first (8) was being written a sixth (9) was suggested. Although we plan to write and present papers on the additional three elements, the DBMS, the Data Bank, and the Data Base Administrator, we have not yet done so, and a seventh paper has already been suggested, regarding the planning and implementation for the ICDB concept. Consequently, many obvious questions (questions that we have been thinking about and on which we have begun to formulate our thinking) have not been addressed. The work is far from complete. It is hoped that the series can be completed in the near future, and all comments are welcome on the work that has been completed, as well as suggestions for that yet to come.

REFERENCES

1. Carbonell, J. R., "On Man-Computer Interaction: A Model and Some Related Issues," *IEEE Trans. on Systems Science and Cybernetics*, Vol. SSC-5, No. 1, January 1969, p. 16.
2. Scott, A. E., "Information Systems—How Do We Get from Here to There?," *GUIDE Secretary Distributions*, Vol. *29*, Denver, Colorado, November 1969.
3. Miller, J. G., "Adjusting to Overloads of Information," in *Organizations*, Vol. II, J. Litter (ed.), John Wiley and Sons, 1969.
4. Simon, H. A., *Models of Man*, John Wiley and Sons, 1957.
5. Schroder, H., M. Driver, and S. Strueferts, *Human Information Processing*, Holt, Rinehart and Winston, 1967.
6. Engles, R. W., *A Tutorial on Data Base Organization*, IBM Corporation, San Jose, California, June 1969.
7. Meltzer, H. S., *Data Base Concepts and an Architecture for a Data Base System*, IBM Corporation, San Jose, California, August 1969.
8. Plagman, B. K. and G. P. Altshuler, "A Data Dictionary Directory System Within the Context of an Integrated Corporate Data Base," *AFIPS Conference Proc.*, Vol. 41, Montvale, New Jersey, 1972.
9. Plagman, B. K. and G. P. Altshuler, "An Integrated Corporate Data Base Concept and Its Application," *Proc. 1972 ACM SIGFIDET Workshop on Data Description and Access*, New York, 1972.
10. GUIDE/SHARE Taskforce Report, "Data Base Management System Requirements," *GUIDE Secretary Distribution* (GSD-023), November 1970.
11. Construction Management System Action Group, *Data Management System Requirements*, Miami, Florida, June, 1971.
12. CODASYL Systems Committee, *Data Base Task Group Report, April 1971*, Association for Computing Machinery, New York.
13. CODASYL Systems Committee, *Feature Analysis of Generalized Data Base Management Systems*, Association for Computing Machinery, New York, May 1971.
14. *Trends in Data Management, Parts I and II, EDP Analyzer*, (R. G. Canning, Pub.) May-June 1971, Vista, California.
15. Schubert, R. F., "Basic Concepts in Data Base Management Systems," *Datamation*, Vol. 18, No. 7, July 1972.
16. Simon, H. A., *New Science of Management Decisions*, Harper and Rowe, New York, 1960.
17. Newell, A., J. C. Shaw, and H. A. Simon, "Elements of a Theory of Human Problem Solving," *Psycholog. Rev.*, Vol. 65, May 1958.
18. Gerrity, T. P., "Design of Man Machine Decision Systems: An Application to Portfolio Management," *Sloan Management Rev.*, Winter 1971.
19. Miller, R. B., *Psychology for a Man-Machine Problem Solving System*, TR 00.1246, IBM Corporation, Data Systems Division Development Laboratory, Poughkeepsie, New York, February 1965.
20. Miller, R. B., *Response Time in Man-Computer Conversational Transactions*, TR 00.1660-1, IBM Corporation, Systems Development Division, Poughkeepsie, New York, January 1968.
21. Miller, R. B., *Archetypes in Man-Computer Problem Solving*, TR 00.1909, IBM Corporation, Systems Development Division, Poughkeepsie, New York, August 1969.

22. Miller, R. B., *A Conceptual Primer on Information Systems for Management*, AR-0989-00-POK, IBM Corporation, Systems Development Division, Poughkeepsie, New York, August 1971.
23. Barkin, S. R. and J. A. Lasky, *The Analysis and Design of Man-Machine Decision Systems: A Behavioral Perspective*, Working Paper Series, Management Information Systems Research Center, University of Minnesota (unpublished).
24. Licklider, J. C. R., "Man-Computer Symbiosis," *IRE Trans. on Human Factors in Electronics*, Vol. HFE-1, pp. 4-11, March 1960.
25. Ghosh, S. P., "File Organization: The Consecutive Retrieval Property," *Communs. ACM*, Vol. 15, No. 9, September 1972.
26. GUIDE International Inc., *"User Language System Requirements," GUIDE Secretary Distribution*, 1973.
27. Senko, M. E., *File Organization and Management Information Systems*, IBM Research Laboratories, San Jose, California (unpublished).
28. Montgomery, C. A., "Is Natural Language an Unnatural Query Language?" *Proc. ACM 1972*, pp. 1075-1078, August 1972.

DUCHESS—A high level information system

by BRUCE J. TAYLOR and S. C. LLOYD

Duke University
Durham, North Carolina

The amount of time which has elapsed between Liebritz's first theoretical description of a computing machine and today's commonplace use of digital computers as extensions of Man's intellectual faculties is a little more than three hundred years. This period has witnessed the birth and death of many trends in the art of mechanical computing, some no more than fads and others becoming established as fundamental truths which are now accepted as axioms of computer science. One of the most firmly established of these latter trends is the quest for generality. As early as 1833, Charles Babbage discerned that in the ideal computing machine, the human operator should have completely flexible control not only over the data to be processed, but also over the algorithms.

The history of computing since Babbage has beeen marked by a terrific emphasis upon flexibility. Turing and Von Neuman laid a theoretical basis of generality in control structures. Recent work in macro-modules (Clark and Bell) has started an explosion in the generality even of hardware. On the software scene, the growth of the concept of modular "structured programming" has opened new horizons in the generality of the programming process. However, it is the myriad computer languages that most clearly point out the trend towards generality. Of the hundreds of programming languages which have grown up in the past decade, all but a few have been designed with a delineated scope of applicability. No matter how well such a language covers its particular field, it is forever relegated to that very restricted area. The most important languages of the modern era, FORTRAN, PL/1, ALGOL, and COBOL, are all marked by a common feature: each provides facilities general enough and powerful enough to implement a wide class of algorithms.

Sadly, the field of information processing and data base management has lagged behind these trends. It seems that the field of information processing systems is characterized by a welter of small systems each designed to implement a given complement of algorithms (such as storage, retrieval, and cross-tabulations). While this set of algorithms is often very large, and the designers may have anticipated many of the needs of the target installations, still we await the advent of a data base management which satisfies Babbage's ideal of flexible control.

The DUCHESS project marks an attempt to design an information management system containing facilities for implementing a wide range of data manipulation algorithms. To this end, we have taken steps to generalize the three basic subdivisions of an information system: the data base, the control structure, and the operating system which supports the system. The primary emphasis in the following discussion will lie upon the desirability and practicality of making every feature of the system as general as possible.

It would be appropriate at this point to present an overview of the DUCHESS system to serve as a reference for discussions to follow. DUCHESS is implemented on a DEC PDP-11 minicomputer with 28 K words of memory. Peripheral resources available to the system include: two RK11 disk drives (2.4 megabytes each), a console teletype, and several CRT-type conversational terminals. It should be mentioned that the DUCHESS concept is not specific to a PDP-11; it could be easily implemented on any machine with similar computational capabilities.

The software side of the DUCHESS system may be divided into four distinct modes: (1) a data base access and management system; (2) a high level programming language with which to implement the data management algorithms. This language provides facilities for interfacing with the data base access module; (3) a complement of user service routines which implement often-used system functions; (4) a multi-user executive system which multiplexes the resources outlined above between multiple users, each in control of one of the CRT terminals. Because the length of this paper is limited, only the first two modules will be discussed. The user routines and the executive are essentially transparent to the user and do not significantly relate to the concept of generality which this paper illustrates.

DATA BASE—DATA DEFINITION LANGUAGE

The first question facing the designer of any data management system is "What data will the system need to record?" In the case of DUCHESS the answer to this question came easily: we would provide facilities for recording any type of data which can be transcribed into a computer-readable format. The next step was to select a suitable set of datatypes for the representation of data. The following set was chosen as adequate to record any type of data:

DATATYPE	DESCRIPTION	INTERNAL STORAGE FORMAT
1. CODE (n)	This type is used to record a selection from within a set of n mutually exclusive choices. Typically, "multiple choice" data is recorded in this format.	log n contiguous bits
2. BOOLEAN	A special case of the CODE datatype used to record yes/no data. The only permissible values are 0 (false) and 1 (true).	1 bit
3. FIXED (d, p)	A fixed point number of d digits with p digits to the right of the implied decimal point. The default for d and p is 6 and 0 respectively.	1 or 2 words (single or double precision is chosen by the values of d and p). Decimal Point location is implicit.
4. FLOAT	A floating-point number of standard PDP-11 floating point format	2 words mantissa 1 word exponent
5. DATE	This datatype is used to record dates (a frequently used datatype) in a compact format.	1 word The date is compressed upon storage and re-expanded at retrieval.
6. TIME	Like the DATE datatype, this datatype records often-used sort of data in a compact format.	1 word. Time is expressed as hours, minutes and seconds since midnight.
7. TEXT (n)	This type is used to record character strings known length (as, for example, a set of three initials). Character strings are truncated or padded with blanks to fit the storage field.	n consecutive bytes ($0<n<256$)
8. TEXT	This is the general case of the TEXT datatype used to record varying length character strings, such as names and addresses with no wastage	1 word pointer into a "free text buffer." Within the free text buffer-one byte for each plus one length byte ($0<\text{length}<256$)

It is immediately evident that, by recognizing many different datatypes and choosing the most efficient representation for each, DUCHESS can gain a considerable savings in data storage space over other systems. Consider, for instance, a certain item of data which may take on any one of four different states. It is evident that the storage of this datum should take no more than two bits to record the alternatives. However, most data management systems allocate space for non-text nodes on word boundaries. This means that, on a 16-bit machine like the PDP-11, 14 bits are wasted. In a data base of ten thousand records, this wastage amounts to more than 8.5 K words of wasted storage space which could have been reclaimed if an appropriate data type (like CODE(n)) with allocation on bit boundaries, had been available. DUCHESS solves the problem of data alignment within records by aggregating all bit-aligned variables into one field, all byte-aligned variables into another field, and all word-aligned variables into a third field. In this way, only one "filler" space is required per field.

It was decided that DUCHESS needed facilities to record both fixed and variable length character strings. Fixed length strings are stored in the most obvious manner: the appropriate number of contiguous bytes are allocated to the string. The length byte which normally precedes a DUCHESS character string is not carried in the record because the length of the string is bound via the data definition facilities. The fact that the location of a fixed-length character string is always known as an offset provides for rapid access via simple address arithmetic. For variable-length strings, the process is somewhat more complex. One word of storage is allocated in the fixed length section of the record for a pointer into free text buffers which are allocated on demand. Thus, it takes two references to access a variable-length text string stored in a DUCHESS record.

The most natural format devised for a data base representation is that of a multi-level hierarchy. It speaks well of the information management field that this format has been almost universally adopted. Human beings tend to look at a large mass of data (like that represented in a typical data base) in terms of its logical subdivisions. Hence, the structural hierarchy fits very naturally into a human's conception of the information recorded in a data base. DUCHESS, like other hierarchical information systems, faces a significant problem in converting a human's data base definition specifications (called the "hierarchy document") into a compact model of the format of a data record.

The user must describe the data to be recorded by preparing a "hierarchy document" for the system. The format of this document is very similar to the format of a PL/1 structure. As an example, we will consider the following portion of a hierarchy:

```
1 BOOK SEGMENT
  2 AUTHOR SEGMENT
    3 NAME
      4 FIRST TEXT
      4 LAST TEXT
      4 INITIAL TEXT (1)
  2 REVIEWER SEGMENT
    3 NAME
      4 FIRST TEXT
      4 LAST TEXT
      4 INITIAL TEXT (1)
```

This document is used as input into a "hierarchy compiler" which constructs a model of the record format for the data base. This compiled version is used to create and initialize the data base file and in compiling application programs.

The above example illustrates two forms of data partitioning under DUCHESS. The simplest form is illustrated by the level 3 node called NAME. Note that this node serves purely as a structuring device and carries no data whatsoever. However, one bit is allocated for each such node in the record. This bit is initially clear and is set whenever one of the nodes below it is given a value. This node type acts as a BOOLEAN node, so that it is possible to poll a node such as NAME and determine whether or not anyone of its sub-items has been altered from its initial state. The assignment of a value to a variable automatically invokes the mechanisms to post such nodes, a process called upward spreading. Of course, this spreading action will continue upward as far as necessary, each spreading operation triggering a spread to the next higher level until a previously posted level or the root node is reached.

The above example also illustrates the use of the segmented data structure in the DUCHESS data base. This concept of a data-dependent data structure is one of the most powerful features of the DUCHESS system and deserves somewhat of an introduction. One of the most significant problems facing a user of a data base system is the tradeoff between his desire to provide space for all possible data and the knowledge that missing data wastes the space allocated for it. The ideal solution to this problem would be to provide a structure wherein no space is allocated for the storage of a datum until it is actually needed. With such a system, the developer could feel free to include even the smallest item of data in his hierarchy structure, knowing that no space will be used unless there is actually data to be recorded. Several information systems have adopted such philosophies with varying degrees of success. The most common technique is to sequentially allocate space within the record value, as it is stored, is prefixed by a node identifier. At retrieval time, the entire record is searched sequentially for the desired node identifier. If the identifier is found, the accompanying data is read back. Certainly, this technique solves the problem mentioned and may be acceptable for use in an inherently sequential medium such as magnetic tape. However, the sequential nature of the search required completely negates the advantages gained in using a high-speed random access device like a disk. Furthermore, the space allotted for the node identifiers often incurs significant overhead as opposed to fixed field organizations. Clearly, "there's gotta be a better way."

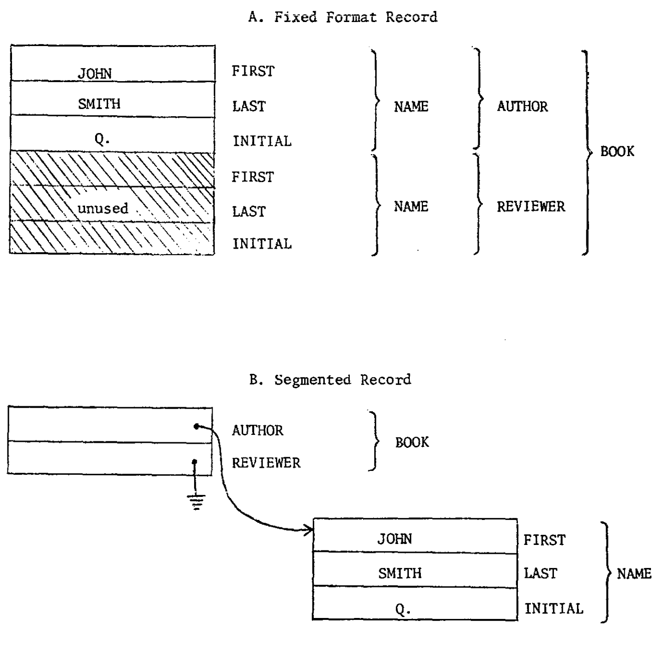


Figure 1—Fixed vs. segmented record format

Data segmentation is an attempt to strike a workable balance between complete freedom in hierarchy planning and the advantages of associating fixed offsets with node information. To this end, there is another DUCHESS datatype not mentioned in the preceding list, the SEGMENT type. To the person generating the hierarchy, the use of the SEGMENT datatype signifies that all nodes immediately subordinate to the segment node are considered as a single logical entity. In terms of our example, assume for a moment that the REVIEWER substructure is not used in every record. By coding REVIEWER with the datatype SEGMENT, the developer indicates that space is not to be allocated in the record for the REVIEWER substructure until one of the terminal sub-fields (FIRST, LAST, or INITIAL) are assigned a value. When the value of any of these fields is recorded in the record, an appropriate amount of space is allocated for the entire segment containing the target node. The allocated space is then linked to its "parent" segment in the record so that later access to the data can descend from the father into the newly allocated segment. It may be useful here to present a visual comparison between the "standard" storage technique and the DUCHESS segment structure. In our example, assume that the only information recorded is the name of the author, JOHN Q. SMITH. The REVIEWER substructure is still in the initial state. Compare the two storage structures diagrammed in Figure 1. For purposes of illustration, the structures shown above have been considerably simplified. Nonetheless, the substantial savings in unused space within the record should be immediately evident. The DUCHESS structure uses one word for a null pointer to represent the missing REVIEWER structure (always with the option of filling in the structure by allocating space for the segment and establishing the REVIEWER link in the BOOK segment). By contrast, the fixed record format requires many bytes of unused space, simply to maintain the record structure. Thus, an infrequently used item costs the developer considerably less if he uses the DUCHESS system of allocation. We feel that this fact contributes significantly to the flexibility of the system.

Because the tradeoff of data to be recorded versus storage used is so important to the data base developer, DUCHESS provides a facility similar to the sequential search method mentioned earlier. Each segment of the hierarchy may contain certain nodes labeled SPARSE. Such nodes generally represent data which will only rarely be stored in the record. Associated with each segment is a variable sized "sparse buffer" to contain the values of all the SPARSE nodes declared within the segment. When none of the SPARSE nodes is assigned, the sparse buffer is not allocated and takes up no storage. Upon assignment of a SPARSE node, a sparse buffer is allocated within the record and linked to the parent segment. Within this sparse buffer, the data of the SPARSE node is recorded, preceded by a node identifier. At retrieval time, the sparse buffer is sequentially searched for the node identifier requested. This process seems identical with the sequential technique mentioned earlier. However, since SEGMENT nodes can be declared as SPARSE, the DUCHESS implementation of this data structure actually represents a mixture of fixed and variable record structure.

At this point, we will anticipate some objections by admitting that the preceding paragraphs glossed over some serious problems raised by the new structure. In a fixed record format system, each datum has a certain offset from the beginning of the record and is instantly accessible by a simple address calculation. In the DUCHESS segmentation system, each item of data is associated with an offset from the base of its segment. The process of locating the segment is mainly one of following a linked list through the data base file. We admit that this is a potentially horrendous task. However, we have succeeded in building enough "intelligence" into the system so that nearly all of the list searching can be obviated and the remaining traversal will not seriously detract from system performance. Once again, we feel that the flexibility gained by implementing this more sophisticated record structure is well worth the extra processing involved. The technique of avoiding the inherent segment lookup is discussed in the next section on the DUCHESS programming language.

DUCHESS PROGRAMMING LANGUAGE

We have already described in part our effort to produce a powerful and flexible data base structure. We believe that this data structure can efficiently record any sort of data which can be transcribed into the computer's input devices. However, this structure would be virtually worthless without an equal degree of freedom in the facilities available for manipulating the recorded data. As mentioned earlier, most data management systems provide a stock set of algorithms for data manipulation. However extensive this set may be, it cannot possibly cover all the needs of an installation which makes frequent use of a data base. A decision was made early in the DUCHESS development to implement a high-level language with two central features: (1) a set of commands designed specifically to interact naturally with the DUCHESS data base system; and (2) an instruction set of sufficient generality that any data-processing algorithm could be realized without having to "strain" either the language or the data base. It order to stay close to a familiar model, we decided to pattern our language after PL/1. The primary difference between the DUCHESS language and PL/1 lies in three areas: (1) the DUCHESS language is a subset of PL/1; we saw no need to implement the more esoteric features of full PL/1. (2) A full set of CRT terminal input/output commands has been added to the GET/PUT reperoire to take advantage of the conversational abilities of a video terminal. (3) The variable reference structure of PL/1 has been supplanted by one which interfaces more naturally with the DUCHESS data base structure. It is this third factor which is discussed in the remainder of the paper.

Recall the objections to the operational characteristic of the DUCHESS segmented data structure raised in the last chapter: the process of looking up a segment through the segment pointers is a slow operation and costly in storage accesses. If a "segment lookup" was required for each reference to a node in the data base, the system would be impossibly slow. However, it is only necessary to perform the "segment lookup" on the initial reference to a segment. The use of a concrete example might serve to clarify the process of "descending" through the segments. Consider the process of setting the name of the author to JOHN Q. SMITH. The requisite DUCHESS statements are:

```
AUTHOR.NAME.FIRST='JOHN';
AUTHOR.NAME.LAST='SMITH';
AUTHOR.NAME.INITIAL='Q'
```

The actions performed are as follows:

(1) Look up the BOOK segment
(2) Look up AUTHOR segment within the BOOK segment
(3) Assign NAME.FIRST
(4) Assign NAME.LAST
(5) Assign NAME.MIDDLE

Notice that only two "segment lookup" operations are necessary to locate the segment AUTHOR. Once this segment is located, all operations within the segment become automatic since the segment base is known. To highlight another feature of this process, consider the job of setting the name of the reviewer to JAMES H. DOE after having carried out the process above:

(1) Discard the base of the AUTHOR segment

(2) Look up the REVIEWER segment within the BOOK segment
(3) Assign NAME.FIRST
(4) Assign NAME.LAST
(5) Assign NAME.INITIAL

Because the segment BOOK had been located in the process preceding this one, the base address of the BOOK segment was already known and hence it was unnecessary to search for it again.

This process of remembering the locations of segments in the "active reference path" is known as "locality management." A locality is the address within the file of a given segment of a given record and the addresses within the record of all its "father" segments. A locality behaves much like a stack: when a program references a segment "deeper" in the hierarchy then the last reference, the top segment of the locality is used as a starting site for the segment lookup, thus short-circuiting part of the lookup procedure by using data already on the stack. Similarly, when a reference is made to a node "higher" in the hierarchy or down a different branch of the hierarchy tree, segments are "popped" off the locality stack until either the target segment is the top one on the locality stack or until the locality stack is in an appropriate state to begin the lookup procedure (as in the preceding example). In order not to overwork a single locality stack, DUCHESS can maintain up to 256 separate active localities. Thus if an applications program needs to reference data from several different segments concurrently, each of the segments can be included in the set of active localities. In these ways, proper locality management in the DUCHESS language removes the primary objection to use of the segmented data structures.

We have described a system of locality management that lets the programmer move around efficiently within the hierarchy structure. However, we find that this new freedom imposes tremendous responsibilities upon the programmer to set up the localities properly before attempting to access a node of the file. Clearly, this new expansion of responsibilities is unacceptable. To confront this new problem, we introduced an element of "intelligence" into the system by placing the bulk of the responsibility of locality management upon the DUCHESS compiler. The compiler accepts as input not only the user's program but also the processed versions of the hierarchy documents for all files to be accessed in the program. Through use of these documents, the compiler has an intimate knowledge of the structure of the data base. When the compiler is called upon to generate code to access a node of the data base, it can check the set of current localities open (even at compile time) for a locality which will provide the appropriate segment base. If no such locality is available, code is generated to establish a locality stack for such use. Although the user retains the facilities for explicit locality control (and he may exercise explicit control over key localities in order to "fine tune" system operation), the compiler has the facilities for carrying out all the "dirty work" of locality management.

In order to ensure that the compiler has enough information to properly manage the localities, it is important that the user write code which follows the stack nature of locality growth. That is, any references to a node in the data base should be preceded by the appropriate locality management code (generated either by the compiler or by the user). At compile time it must be possible to exhaustively enumerate the paths the program control might take and, in the code for each path, generate the appropriate locality management code. The major obstacle to such analysis is the abuse of the GOTO statement. The target statement of a GOTO has two distinct entry paths, one by natural program flow and the other by the action of the GOTO statement. Each of these entry locations requires a different sort of locality management to bring the active localities into the state needed by the target statement. In the case where one statement is the target of multiple GOTO statements, the problem becomes, for all intents and purposes, insoluble. Apparently, we need a mechanism for insuring that each DUCHESS statement has exactly one entry path: the natural program flow. The mechanism chosen is distressingly simple; the DUCHESS compiler prohibits any form of the GOTO statement. To assume the function of this control statement, a set of control structures have been implemented, providing such facilities as bypassing a group of instructions, iterative looping, looping on condition, and a form of multi-way branching. The basis of these control structures results from current investigations in 'structured programming.' It has been shown that not only can the task of compilation be made more efficient and object code be optimized but surprisingly, programs written with such restraints are easier to debug. We anticipate that new programmers, after an initial period of adjustment, will adapt well to a new sort of control structure.

In this paper (which, in retrospect, seems woefully inadequate to convey the new ideas generated in the course of the DUCHESS design) we have tried to illustrate that the primary problem with most modern information systems lies in an inherent rigidity both in their data structure and in the control structure. This ridigity tends to lead their users into assuming that the scope of information management systems should include only those functions presently implemented: storing, retrieval, searching and sorting, and elementary analysis. The ideal information system should have these capabilities plus opportunities for developing individualized procedures, via a programming language or other means. We hope that the DUCHESS project will illustrate that the practical implementation of such a flexible system is not only possible but no more difficult that the design and implementation of a rigidly restricted system. Furthermore, we demonstrate that this flexibility does not come with a higher price tag, for DUCHESS efficiency compares favorably with its less flexible predessors.

REFERENCES

1. Dean, A. L. (editor), *Proceedings of 1972 ACM-SIGFIDET Workshop Data Description, Access, and Control,* (available from the ACM).

2. Gates and Poplawski, "A Simple Technique for Structured Variable Lookup," *Communications of the ACM*, Vol. 16, No. 9, September 1973, pp. 561-564.
3. Jones *et al.*, *CODASYL Data Base Task Group Report*, April 1971, (available from the ACM).
4. McKeenan, Horning, and Wortman, *A Compiler Generator*, copyright 1970, Prentice-Hall Inc., Englewood Cliffs, N.J.
5. Ornstein, Stucki, and Clark, "A Functional Description of Macromodules," *Proceedings of the Spring Joint Computer Conference, 1967*, p. 337.

An analytical model for information processing systems*

by SHENG-CHAO HUANG

Sperry Gryoscope
Great Neck, New York

and

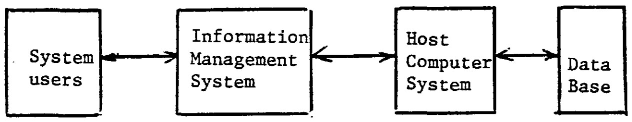
AMRIT L. GOEL

Syracuse University
Syracuse, New York

INTRODUCTION

Information processing systems are an important sector in the application of computer technology to the fields of on-line data processing and the management of large complicated data bases.[1] With an increase in the number of systems being designed and used, there has been an increasing emphasis on the modeling and performance evaluation of such systems. In Reference 4 Nunamaker presented a procedure for the design and optimization methodology of such systems. In some cases,[5,6] the evaluation procedure was reduced to the evaluation of file structure and data base organization. Kobayashi[7] gave an algebraic modeling of information structures. In this paper, we carry out the study of mean response time by taking into consideration host processor environment, user characteristics and the data base structure. Total system response time is the main goal of this investigation.

An information processing system is defined here as a total operating environment consisting of users; information management system, which includes jobs and procedure, data definition and other software packages; host computer system; and data base structures. The interrelationship among elements of this total system is shown in the following block diagram.



In general, the information management system can be considered as a special purpose operating system consisting of two parts. The first part is called the main system which has the capability of file creation, data maintenance, job and procedure creation, etc. The second part includes subsystems which could be treated as the internal service system for data access, data index, storage allocation, privacy protection, restart and recovery and other miscellaneous functions.

The information management system interfaces with its host processor through the host operating system. The data base is stored in the memory hierarchical structure of the host system. Thus, a user can reach his data only through the path shown in the above diagram.

Total system model parameters to be included in the sequel are (1) user characteristics, (2) overhead time from the information management system and the host processor, (3) host processor operating environment which would determine the waiting time of a job and the main memory space allocated for the job execution, and (4) the data base structure which is reflected in the logic data relationship and the physical data storage.

In the next section, we briefly present an algebraic and topological model of the data structure within an information processing system.

DATA BASE STRUCTURE AND DISTANCE MATRICES

Let E be the set of entities to be considered within an information processing environment. Let

$$\Pi = (\Pi_1 \ldots, \Pi_n)$$

be the distinct properties assigned to the elements of E. Then Π is to be treated as a mapping from the entity space E to the property space V, i.e.,

$$\Pi: \quad E \rightarrow V = V_1 \times V_2 \times \cdots \times V_n.$$

The point $\Pi(e) \in V$ is then called the record of $e \in E$. A subset D of V such that $D = \{\Pi(e); e \in E\}$ is called the data base of the entity E. If

$$D = F_1 \cup F_2 \cup \cdots \cup F_r \quad \text{and} \quad F_i \cap F_j = \phi \quad \text{for} \quad i \neq j,$$

then F_i is called a data file in D, where ϕ is an empty set.

From now on, we shall use x_i, $i = 1, \ldots, N$, to denote a record in D or F, i.e., $x_i = \Pi(e_i)$, $e_i \in E$.

For every x_i in D, a unique physical space is assigned in the secondary storage device. Let S indicate the physical storage space. Then the address assignment can be considered

* This research was supported, in part, by RADC, Rome under contract number F-30602-72-C-0281.

as a one-to-one mapping α from D to S, i.e.,

$$\alpha:\ D \rightarrow S. \tag{1}$$

An essential element of information processing is data retrieval which could be treated in two parts. One part is the mapping α^{-1} from S to D, and the other part is the data transmission from the secondary storage device to the main memory. One such operation is to find and retrieve all records within the data base which possess certain property values. Explicitly, let G be a subset of a data file F, such that

$$G = \{x : x \in F,\ \rho(x) = T\} \tag{2}$$

where ρ is a logic function and T is defined as logical true. This implies a continual data retrieval within a system. For example, the data retrieval for ρ is to find all records in F such that ρ is true. To start with, the first record has to be retrieved and after that other records will be retrieved sequentially according to the topological relationship between these records. In the next paragraph this topological relationship is defined probabilistically.

Let $x_1, x_2, \ldots, x_N$ be all records within a system which are of interest to us. In other words, $x_1, \ldots, x_N$ could be all records in a data base or in a certain data file. Suppose for an arbitrary retrieval function ρ_r, we want to obtain records among $x_1, \ldots, x_N$. We define a probability matrix $P^{(r)}$ of ρ_r with its elements for $i, j = 1, \ldots, N$, being

$P_{0i}{}^r$ = probability that the first record to be obtained is record x_i,

$P_{ij}{}^r$ = probability that x_j is demanded immediately after the retrieval of x_i,

and

$P_{ie}{}^r$ = probability of reaching the end of the process immediately after the retrieval of x_i.

Note that there are many different jobs to be processed and the number of occurrences of processing for a job vary from one to another. Assume that there are M application programs and M retrieval functions, $\rho_1, \ldots, \rho_M$. Furthermore, assume that the probability of occurrence of ρ_r is p_r such that $\sum_{r=1}^{M} p_r = 1$. Then we can define a matrix P

$$P = \begin{pmatrix} P_{01} & P_{02} \ldots P_{0N} & P_{0e} \\ P_{11} & P_{12} \ldots P_{1N} & P_{1e} \\ \vdots & & \\ P_{N1} & P_{N2} \ldots P_{NN} & P_{Ne} \end{pmatrix}, \tag{3}$$

whose elements are defined as follows

$$P_{0i} = \sum_{r=1}^{M} p_r p_{0i}{}^r, \qquad i = 1, \ldots, N; \quad p_{0e} = 0,$$

$$P_{ij} = \sum_{r=1}^{M} p_r p_{ij}{}^r, \qquad i, j = 1, \ldots, N, \tag{4}$$

and

$$P_{ie} = \sum_{r=1}^{M} p_r p_{ie}{}^r, \qquad i = 1, \ldots, N.$$

In practice, p_r, $r = 1, \ldots, M$, and hence P is a function of time. However, for simplicity, we shall assume that p_r's are constants and hence P is a constant matrix. Note that, for $i = 0, 1, \ldots, N$, it can be easily shown that

$$\sum_{j=1}^{N} p_{ij} + p_{ie} = 1 \tag{5}$$

The matrix P defined above is called the "logical distance matrix" or the "reference matrix." It represents a pattern of reference behavior in a particular system and can be considered as the measurement of logical distance among elements within a data file or a data base. It is apparent that the elements of P are the parameters determined by the user's characteristics and differ from one installation to another.

The mapping α defined in (1) assigns a physical space to every x in D. This should be done is such a way that the preservation of distance is obtained as much as possible. The physical distance, t_{ij}, between x_i and x_j is defined as the time required for the retrieval of x_j immediately after the retrieval of x_i. It is not necessarily true that $t_{ij} = t_{ji}$. Note that t_{ij} is a random variable and let $E(t_{ij})$ denote the expected value of t_{ij}. Thus we can define the "physical distance matrix" in a data base as follows

$$E(T) = \begin{pmatrix} E(t_{01}) & E(t_{02}) \ldots E(t_{0N}) & E(t_{0e}) \\ E(t_{11}) & E(t_{12}) \ldots E(t_{1N}) & E(t_{1e}) \\ \vdots & \vdots & \vdots \\ E(t_{N1}) & E(t_{N2}) \ldots E(t_{NN}) & E(t_{Ne}) \end{pmatrix}, \tag{6}$$

where

$E(t_{0j})$ = the mean time to obtain x_j from the secondary memory to the main memory when x_j is the first record in the string of reference and

$E(t_{0e}) = E(t_{1e}) = \cdots = E(t_{Ne}) = 0.$

The distance matrix defined in (6) is not only dependent on the physical file organization of the data base in the memory hierarchy, but is also dependent on the search and indexing methods within an information management system.

THE MEAN RETRIEVAL TIME

From the previous analysis it can be seen that the following quantity T' can be used for the evaluation and optimum design of physical storage assignment,

$$T' = \sum_{i=0}^{N} \sum_{j=1}^{N} p_{ij} E(t_{ij}), \tag{7}$$

where N is defined as before. Obviously, it is desirable to keep T' as small as possible. In order to do so, one problem in the address assignment is to minimize (7) with respect to $E(t_{ij})$, $i, j = 1, \ldots, N$, subject to certain appropriate constraints. Because p_{ij}'s are determined by the user's characteristics, they are to be treated as constants in the minimization procedure. Since it is not the main concern of the paper at this point, we shall go back to the modeling aspect of the average retrieval time.

The model given in (7) does not take into consideration the fact that in certain jobs the string of references may be quite long and some records may be referred to more than once during the execution of a job. This means that, for a fixed amount of main memory allocation, a record may already be in residence when it is referred to so that a search in the secondary memory space is unnecessary. Now assume that the main memory space allocated for the execution of an information processing job be m page frames (or blocks). For a given job r, define a matrix $Q^r=(q_{ij}^r)$, $i,j=1,\ldots,N$, so that q_{ij}^r is the probability of x_j being in residence immediately after the retrieval of x_i. Thus a matrix for missing records probability in the execution of job r could be defined as $[I-Q^r]=(1-q_{ij}^r)$ where I is defined as a matrix with all entries equal to 1. It can be readily seen that the searching probability of record x_j immediately after the retrieval of x_i is given by $p_{ij}^r(1-q_{ij}^r)$. This naturally leads to the definition of "search matrix" A_r, as follows:

$$A_r=\begin{pmatrix} p_{01}^r & p_{02}^r & \ldots p_{0N}^r & p_{0e}^r \\ p_{11}^r(1-q_{11}^r) & p_{12}^r(1-q_{12}^r) & \ldots p_{1N}^r(1-q_{1N}^r) & p_{1e}^r \\ \vdots & & & \\ p_{N1}^r(1-q_{N1}^r) & p_{N2}^r(1-q_{N2}^r) & \ldots p_{NN}^r(1-q_{NN}^r) & p_{Ne}^r \end{pmatrix} \quad (8)$$

Note that Q^r matrix should be different for different jobs. However, its probability average can be obtained in the same way as we obtained the matrix P in the previous section. It should be pointed out that since q_{ij}^r's are functions of time, the search matrix A_r is also dependent upon the duration of a job being in the execution stage.

Suppose a job r is being processed and it keeps a string of records as reference, i.e., let $K=\{x_1,\ldots,x_{k_r}\}$ be the set of records referred to during the processing of a job. Then the total time spent in data retrieval is given by

$$T_r=E(t_0^r)+\sum_{i=1}^{N}\sum_{j=1}^{N}p_{ij}^r(1-q_{ij}^r)E(t_{ij}) \quad (9)$$

where $E(t_0^r)$ is the expected retrieval time for the first record. In general, q_{ij}^r depends upon p_{ij}^r as well as on the time sequence. However, for simplicity, we here assume that

$$q_{ij}^r=\begin{cases}\dfrac{m}{k_r}, & \text{for } k_r>m \\ 0, & \text{for } k_r\le m\end{cases} \quad (10)$$

Equation (10) is based on the assumption that the system has reached its steady state where all the allocated main memory space has been occupied. Thus, this part of analysis could be considered as a steady state analysis. The transient case, which deals with the time interval between the starting of a job and the time when all the main memory space is occupied, shall be considered in another paper. Assuming $k_r>m$ and substituting (10) into (9), we obtain

$$T_r=E(t_0^r)+\sum_{i=1}^{N}\sum_{j=1}^{N}p_{ij}^r\left(1-\frac{m}{k_r}\right)E(t_{ij}) \quad (11)$$

Note that parameters k_r and p_{ij}^r are predetermined by the application programs and jobs. Hence, all of them should be considered part of the user's characteristics. The value of m is hardware dependent and $E(t_{ij})$ are characteristics of physical file organization and search mechanism. Thus, equation (11) gives the total retrieval time for a job r in terms of the user's characteristics, physical file organization, searching methods and the hardware constraints.

If the total number of jobs within an information processing system is M and if each job r has the probability p_r of being processed, then the expected value of the total retrieval time for a job is

$$E(T_R)=\sum_{r=1}^{M}p_r\left\{E(t_0^r)+\sum_{i=1}^{N}\sum_{j=1}^{N}p_{ij}^r\left(1-\frac{m}{k_r}\right)E(t_{ij})\right\} \quad (12)$$

Equation (12) can be further simplified by letting

$$E(t_0^r)=\frac{1}{N}\sum_{i=1}^{N}E(t_{0i})=E(t_0), \qquad \text{for} \quad r=1,\ldots,M, \quad (13)$$

and

$$\sum_{r=1}^{M}p_r p_{ij}^r=p_{ij}; \qquad \sum_{r=1}^{M}\frac{p_r\cdot p_{ij}^r}{k_r}=\frac{p_{ij}}{\bar{k}} \quad (14)$$

By substituting (13) and (14) into (12), we have

$$E(T_R)=E(t_0)+\sum_{i=1}^{N}\sum_{j=1}^{N}p_{ij}\left(1-\frac{m}{\bar{k}}\right)E(t_{ij}) \quad (15)$$

Note that $\bar{k}$ and p_{ij} are the parameters inherent within the set of all possible user's jobs and thus represent a measure of the user's characteristics.

It needs to be noted that $E(t_{ij})$ consists of two major parts; one part is the search of data address and the other part is to get the data from specific address in the secondary memory to the main memory. The first part depends on the search method and the way files are organized, while the second part depends upon the way data are stored on the secondary storage device. This means that both the search method and the file organization are the parameters in equation (15).

TOTAL SYSTEM RESPONSE TIME

The total system response time is defined as the interval between the time a user submits his job and the time when the satisfactory result is obtained. Suppose that the host system is multiprogrammed and a job presented to it will be forced to spend some time in the system queue. Thus, the total response time could be divided into two portions: waiting time T_w and processing time T_p. Since it is not our intention to investigate the queueing properties of the host system, we simply assume that the mean waiting time is $\bar{T}_w$ under the operating environment. Then the total system response time, T, could be expressed as

$$T=\bar{T}_w+T_p \quad (16)$$

The time spent in job processing could be viewed as the sum of the time for information fetching, time for job execution which includes decoding and binding, etc., and the overhead time which will include input-output overhead as well as system overhead. Let T_R, T_e and T_0 denote the times spent in data fetching, the time for job execution and the overhead time, respectively. Then T_p can be written as

$$T_p = T_R + T_e + T_0, \tag{17}$$

or

$$E(T_p) = E(T_R) + E(T_e) + E(T_0). \tag{18}$$

In some information management systems the job execution is divided into two stages. In the first stage a job is prepared by the management system before transfer to the host operating system for real execution. The procedure at this stage of work could include data search and retrieval, data binding, data decoding, etc. During the second stage of processing, the prepared jobs are executed through the host operating system. Under such a circumstance, $E(T_e)$ could be further divided into two terms:

$$E(T_e) = E(T_{em}) + E(T_{eh}), \tag{19}$$

where $E(T_{em})$ is the mean time for the pre-execution of a job which is under the control of the information management subsystem and $E(T_{eh})$ is the mean execution time under the supervision of the host operating system. Likewise, the overhead time can also be classified into two categories as follows:

$$E(T_0) = E(T_{0m}) + E(T_{0h}) \tag{20}$$

where T_{0m} is due to the management subsystem and T_{0h} is the responsibility of the host operating system.

Assume that the overhead factor is approximately equal for both the information management system and the host operating system. Then we can write

$$E(T_0) = \delta E(T_p), \tag{21}$$

where δ will be called the overhead coefficient. Substituting (21) into (18), the processing time becomes

$$E(T_p) = \frac{1}{1-\delta}\left[E(T_R) + E(T_e)\right] \tag{22}$$

It then follows from (22), (15) and (16) that the mean response time of an information processing system could be expressed as follows:

$$E(T) = \bar{T}_w + \frac{1}{1-\delta}\left[E(T_R) + E(T_e)\right]$$

$$= \bar{T}_w + \frac{1}{1-\delta}\left[E(t_0) + \sum_{i=1}^{N}\sum_{j=1}^{N} p_{ij}\left(1-\frac{m}{\bar{k}}\right)E(t_{ij}) + E(T_e)\right] \tag{23}$$

In some cases T_e can be assumed to be linear with respect to the task length, i.e., for some constant C,

$$E(T_e) = C\bar{k}.$$

Then (23) could be rewritten as

$$E(T) = \bar{T}_w + \frac{1}{(1-\delta)} \times \left[E(t_0) + \sum_{i=1}^{N}\sum_{j=1}^{N} p_{ij}\left(1-\frac{m}{\bar{k}}\right)E(t_{ij}) + C\bar{k}\right] \tag{24}$$

Equations (23) and (24) indicate that the response time of an information processing system is a function of the user's characteristics, host processor operation environment, overhead factor, and the data base structure. The user's characteristics are reflected in the fact that $\bar{k}$ and p_{ij}'s are parameters determined by the jobs and procedures. The average waiting time in the host system's queue and the overhead time could be considered as the host system operating environment and the $E(t_{ij})$'s and p_{ij}'s could be considered as properties of the data base structure.

REMARKS

Since the research reported here is a first step toward analytic modeling and performance evaluation of information processing systems, the results obtained here are by no means final. Further research in this direction is needed. Currently, two research efforts are being pursued; one in the area of model refinement and the other in the area of data procurement and analysis.

REFERENCES

1. Tobias, M. J., and G. M. Booth, "The Future of Remote Information Processing Systems," *Proceedings FJCC 1972*, pp. 1025-1035.
2. Dixon, P. J., and J. Sable, "DM-1: A Generalized Data Management System," *Proceedings SJCC 1967*, pp. 185-198.
3. Fichten, J. P., "The Weyerhaeuser Information Systems—A Progress Report," *Proceedings FJCC 1972*, pp. 1017-1024.
4. Nunamaker, J. F., Jr., "A Methodology for the Design and Optimization of Information Processing Systems," *Proceedings SJCC 1971*, pp. 283-294.
5. Myers, J. E., and S. K. Chooljian, "An Approach to the Development of an Advanced Information Management System," *Proceedings SJCC 1970*, pp. 297-306.
6. Severance, D. G., and A. G. Merten, "Performance Evaluation of File Organizations through Modeling," *Proceedings ACM 1972*, pp. 1061-1072.
7. Kobayshi, I., "An Algebraic Model of Information Structure and Information Processing," *Proceedings ACM 1972*, pp. 1090-1104.

A model for a generalized data access method*

by RANDALL L. FRANK

The University of Utah
Salt Lake City, Utah

and

KOICHI YAMAGUCHI

The University of Michigan
Ann Arbor, Michigan

INTRODUCTION

The proliferation of the methods used in modern operating systems to access data is apparent. In the operating system OS/360 alone there exist multiple ways of accessing sequential data (BSAM, QSAM, BPAM), indexed sequential data (BISAM, QISAM) and directly addressable data (BDAM).[4] If one adds to this the variations of the above used by various systems that run under the control of OS/360, such as IBM's data base management system IMS/360, there exists almost a countless number of ways to access and store data within a computer system.

This proliferation of data access methods has left persons involved in the design, implementation or evaluation of new or existing access methods with an almost hopeless task. In order to better understand the nature of data access methods, a model has been developed in whose terms existing and proposed data access methods can be stated. This paper will discuss the components of such a *generalized data access method* and give examples of its use in modeling existing data access methods. Parts of the model to be presented are at a further state of development than others, and, therefore, at times a formal discussion of parts of the model may be replaced by a more informal functional description.

A discussion of existing data access methods

Since the need for such a generalized data access method is in part based on the fact that we lack a common basis with which to discuss data access methods, it is difficult to offer very many global remarks about existing data access methods. Nevertheless, several general observations about data access methods will be offered.

Computing hardware, in general, tends to be rather inhospitable to the average users of a computing system. In particular, input/output hardware and its associated protocol tend at times even to make system's programmers cringe. Thus, one of the more important functions of data access methods is to provide a cleaner user interface between the user and the underlying hardware.

While the hardware on a given computing system remains fairly constant, user requirements and needs change constantly. Thus, another important function of data access methods is to take a fixed hardware/input-output environment and provide a virtual environment which more nearly matches the environment desired for a particular application. However, any time an operating system designer attempts to provide a fixed number of such virtual applications oriented environments, users will come along whose requirements are not met by the access methods provided. Thus, many operating systems provide for "escape" access methods which allow the user (and, indeed, require him) to interact with the intricacies of the input/output hardware, providing him with only basic support. An example of such a facility is the EXCP (Execute Channel Program) access method in OS/360.[5]

Clearly, it would be desirable to provide for tailored accessing methods without forcing a user to spend undue amounts of time learning the details of a particular hardware system. One purpose of the generalized data access method model to be presented here is to allow for such data access methods tailoring at a fairly high level. Thus, the parameters to such a model must be in the form of user-oriented, and where possible, machine independent languages. The user should be able to state his access method requirements in languages natural to his application, rather than those convenient to a particular machine.

Goals of this research

Several of the goals of this research effort have been alluded to above. Four primary goals to be set forth for this

* Research sponsored in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR-72-2219. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

effort are:

(1) the ability to model existing and proposed access methods.
(2) the ability to compare different data access methods in a common terminology.
(3) to provide a facility with which one can *easily* create and try out new access methods tailored to a particular application.
(4) to enable one to read and write files which are foreign to a particular hardware/software system.

The first two goals above are highly interrelated, and amount to providing a common language with which to discuss data access methods. Attempts to perform comparative analysis of file organizations and access methods (see, for example, Severance[9]) have found that before any analysis could be performed, a common model had to be developed. While models have been developed to facilitate the analysis of specialized aspects of data access methods, for example, performance aspects, little work has been done previously in developing a model with applicability to a wide range of problems.

As was mentioned earlier, when the access methods provided by a vendor are not ideal for a particular application, one is forced to either make do with an existing access method, or design one from the ground up, and specify it usually at the machine level. It is interesting to note that even computer vendors have found their own provided access methods unsuitable for some of the applications software they provide. For example, IBM, in designing IMS/360, decided not to use the standard access methods of OS/360, but instead to develop new ones particular to IMS, such as HISAM.[6] Even programs such as the IBM Mathematical Programming System/360 have ignored operating system provided access methods in favor of specialized accessing techniques using the above mentioned EXCP facility. While these above two major software efforts could afford the time and expense needed to develop new access methods from scratch, the normal user, finding himself in a situation where existing access methods lead to sub-optimal results, often cannot afford the time or money, or lacks the expertise, to design and program a new access method. Thus, a high-level access method design facility is called for.

The fourth goal, which was the original motivation behind this work, has to do with the translation of foreign data files. Foreign in this sense implies software and/or hardware incompatabilities. In the course of research in general file translation at The University of Michigan,[3,1] it became apparent that two of the functional components of a generalized data translator were readers and writers.

Since the basic premise of The University of Michigan research was that data translation could be performed using a very high level stored data description language, it follows that these general readers and writers would likewise have to be driven by statements in this high level language. While the basic design of such general readers and writers is often simpler than that of a completely general data access method, much of the underlying model is the same. For example, in general data translation the question of the data base update function can be ignored while in a general data access method this is a necessary function.

A review of related work

There are several research and/or developmental efforts that are related to our endeavors. However, none of them appear to fully comply with all four of our goals as stated in the previous section. Some of the more interesting research includes work on formulating a model for data description by IBM and by the Stored-Data Definition and Translation Task Group of CODASYL, and a development of a description-driven data translator at The University of Michigan.

Data independent architecture model (DIAM)

A group at IBM research in San Jose has formulated a model for describing data structures, as well as data accessing properties of a wide variety of data base systems.[8] The model, called the Data Independent Architecture Model (DIAM), uses a limited number of basic concepts so that it can be useful not only in describing and comparing various aspects of existing data base systems, but also in designing and implementing new data accessing methods.

The DIAM structure is based upon the four hierarchically related models in describing data, starting at the very abstract view of data in terms of entities and relationships among them, down to the encoding on physical devices. The most interesting features of DIAM are seen in the second and the third levels in the structure, the String (Access Path) Model, and the Encoding Model. In the String Model, various access path structures independent of their encoding techniques and physical implementations are described by varying parameter values of the three types of access path (strings) and by specifying simple operations on these strings.

The stored representation of each element of the access path structure thus defined is depicted in the Encoding Model. All the relevant information on encoding each element of the access path structure is collected in a fundamental unit, called the Basic Encoding Unit (BEU), so that variations in encoding can be nicely expressed in terms of the BEU structure and the process of data factoring applied on BEUs. The combination of these three concepts, access path structures, BEUs, and data factoring, provides DIAM with a general capability of characterizing and evaluating existing and proposed data base systems.

Stored-data definition and translation task group (SDDTTG)

In 1970 a task group of the CODASYL Systems Committee, the Stored-Data Definition and Translation Task Group (SDDTTG), was formed to formulate a stored-data definition language, which is a means of explicitly and formally defining data as they appear on physical media, in order to facilitate data exchange among possibly dissimilar systems. The initial results of the Task Group were reported at the 1970 SIGFIDET (then SICFIDET) Workshop.[11]

Two major criteria have been set forth by the Task Group in developing a model for stored-data description. One is the generality of the model so that it cannot only be applied to existing structures, but also easily be extended to new structures. The other is the separation of logical aspects of stored-data from its physical representation in storage.

In accordance with this guide line, the Task Group introduced a model at the 1972 SIGFIDET Workshop.[12] The model is based on the stored-data description models developed by Smith[10] and by Taylor.[13] The model currently being revised by taking the concepts presented in DIAM into consideration consists of two independent streams of data description. One stream describes logical aspects of stored-data, and the other describes the structure of storage media on which the data is represented. These two descriptions are combined into one in order to complete the total description of stored-data.

Data translation project

The Data Translation Project at The University of Michigan has expressed as its goal the development of a general methodology for transferring data from one environment to another. With this research goal in mind, the project undertook, as its first year's work, the task of developing a prototype data translator, in order to demonstrate the feasibility of the technical approach (i.e., a language-driven data translator as stated in an earlier section), and to gain a better understanding of the problems associated with data translation.[1]

The prototype translator developed translates files generated by NIPS, which operates on an IBM/360 computer, into input files for WWDMS, which operates on a Honeywell H-6000 computer.[2] Although in a rather restricted manner, the major components of the translator are driven by two forms of high level languages: stored-data definition language and translation definition language. The stored-data definition language is used to describe input and output files and the translation definition language is used to make an association between input and output file descriptions. After a successful accomplishment of the first year's task, the project is currently undertaking the task of developing a more general data translator by relaxing the allowable classes of input and output files, as well as by making the translator more language-driven.

OVERVIEW OF THE GENERALIZED DATA ACCESS METHOD MODEL

This section will present and motivate the overall structure of the generalized data access method model. Successive sections will discuss in depth the various components of the model.

The elements of the model to be presented shortly can be divided into two categories, namely language elements and algorithms. The generality of the model is derived from the fact that the algorithms are driven by high-level language descriptions of the desired access method to be modeled. Thus, the algorithms do not as such constitute a particular access method, but merely allow for the simulation of any access method which can be described in the language elements of the model.

A block diagram of the model will be useful for future discussion, and is shown below:

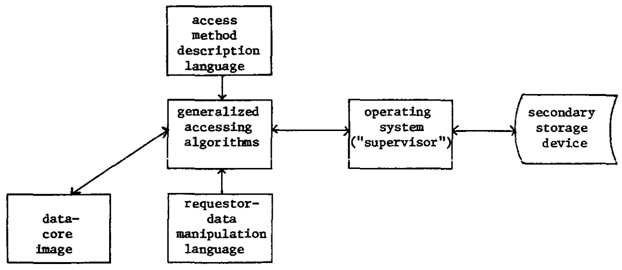

access method description language
generalized accessing algorithms
operating system ("supervisor")
secondary storage device
data-core image
requestor-data manipulation language

Figure 1—A model for a generalized data access method

The main substantial difference between the above model and those of existing specialized access methods is the addition of the access method description language. However, it is this additional element which enables the remaining components to act in a generalized fashion.

The operating system component is placed in the model based upon the realization that such a generalized access method would not be used in a vacuum. However, the operating system here is meant to represent just the minimal environment necessary to support the generalized access method in a multi-user environment. The operating system assumed here excludes components currently thought of as file systems and specialized access methods. These latter components would be built on top of the generalized access method. Thus, the entire model could be considered to be within the realm of a comprehensive operating system.

The "verbs" of the data manipulation language are, of necessity, somewhat dependent on the access method described. For example, access methods which provide for no direct addressing could not (easily) support a data manipulation command requesting that the system deliver a record whose key has a certain value.

However, this is not meant to imply that there is a fixed data manipulation language. The verbs in the data manipulation language are only restricted by the descriptive capabilities of the access method description language and the algorithms which use that language. Thus, one could define, for an inherently sequential access method, a "find direct" verb. However, the semantics for such a verb in a sequential access method might necessitate a linear search of the data base or file.

Certain data manipulation language constructs, such as "next" relative to a current position in a file, are easily implementable for most access methods. However, its implementation may have very different specifications for different structures on secondary storage. The definition of "next" would be quite different for a sequentially organized file as opposed to a list structured file. Thus, while the external semantics of "next" are access method invariant, its implementation may vary widely.

Thus, the data manipulation language is only access method independent to the extent that one specifies not only an access method in the access method description language, but also an implementation for all desired data manipulation commands. Precisely which data manipulation commands should be specified for *all* access methods will not be discussed here, although, through example, certain common data manipulation commands will be presented.

Limitations of the initial research and design effort

In order to limit the scope of the work, several assumptions were made as to the desired level of generality. These limitations have to do primarily with the class of machines usable with a single implementation of the generalized accessing algorithms.

The primary interest here is in studying differing access methods and not in the detailed input/output hardware of computer systems. Therefore, it was decided that description of the access methods would stop at the level above that of the actual input/output instructions of a machine. This restriction limits the usability of an implementation of the generalized accessing algorithms to a given machine architecture. This also implies that the generalized accessing algorithms are responsible for, via "hard code," the translation of access requests in the data manipulation/access method description languages into actual I/O commands for a particular machine.

While many aspects of I/O device architecture are described in the languages of the model, it is assumed that the accessing algorithms know, in a pre-defined sense, how to interpret such descriptions for a given machine architecture. For example, one section of the access method description language discusses the development of secondary storage device addresses. However, the accessing algorithms would be responsible for converting a data request to/from such an address into actual hardware I/O commands. These might include a seek request to a particular cylinder, followed by a transfer request.

An obvious extension of this work would be to provide languages for the description of the structure of the I/O instructions of a machine. Further levels of description might include generalized handling of I/O interrupts and error conditions. While this area is being investigated, it will not be considered further in this paper.

THE ACCESS METHOD DESCRIPTION LANGUAGE

Central to the concept of the generalized data access method is that of high level languages for describing access methods. The access method description language is itself made up of several sub-languages. The two most important ones, to be discussed here, are the device description language, and the accessing language.

Since the accessing language is based upon primitives defined in the device language, the device description language will be presented first.

The device description language

The device description language is based on an assumption about the current nature of secondary storage devices. Namely, that such devices can be described in hierarchical terms, with each element of the hierarchy using only elements of lower levels as its components.

For example, one *possible* hierarchical description of a disk drive consists of storage cells at the lowest level, i.e., positions where bits or characters of data can be placed. At higher levels, records can be viewed as being composed of storage cells, tracks of records, cylinders of tracks, and a disk volume as being composed of cylinders.

It should be noted that this decomposition of a secondary storage device into hierarchical levels is not unique for a particular device, but merely possible. For example, in the above device description, tracks on a single disk surface could be considered to make up the next higher level of the hierarchy. These disk surfaces would then comprise a disk volume.

Thus, the device description language must not contain any assumptions as to the hierarchical breakdown of devices, and must allow for any realistic description. As such, there are no "pre-defined" names for components of devices. Any semantics associated with a device component (such as "record") are given by the user when the accessing of the device is described in the accessing language. The accessing language description is done in terms of the device components specified in the device description language.

Rather than present detailed syntactic and semantic descriptions of statements in the language, a functional description of the language will be presented, followed by possible examples of its use. The language is at a state of development where detailed syntactic decisions have not been completely specified. The examples, therefore, are meant to merely give the flavor of the language.

Elements of the device description language

The two major functions of the device description language are to define the aforementioned device hierarchy, and to describe the scheme used in addressing the device.

As an example device, the IBM 2314 disk drive will be used.[7] In order to minimize the complexity of the example, many low level details associated with such a device will be ignored. Included in this category are problems such as damaged track demarcation and alternate track selection. In a working system based on this model, problems like this would necessarily have to be described and handled.

The device hierarchy is described from the "bottom up" so as to allow for components of one level to be used in the description of higher levels. Keywords in language are underlined. The syntax here is merely illustrative, and should not be viewed as a formal specification.

The first section of the language defines the device hierarchy, and gives attributes of each level of the hierarchy.

```
DEVICE IBM-2314 DESCRIPTION;
    STORAGE CELL IS BYTE;
        BYTE CONTAINS 8 BITS;
    COMPONENT IS RECORD;
        RECORD COMPONENTS ARE BYTE;
        LENGTH OF RECORD IS MIN 1 BYTE
          MAX 7294 BYTE;
    COMPONENT IS TRACK;
        TRACK COMPONENTS ARE RECORD;
        LENGTH OF TRACK IS MIN 1 RECORD
          MAX 71 RECORD AND MAX 7294 BYTE;
    COMPONENT IS CYLINDER;
        CYLINDER COMPONENTS ARE TRACK;
        LENGTH OF CYLINDER IS 20 TRACK;
    COMPONENT IS IBM-2314;
        IBM-2314 COMPONENTS ARE
          CYLINDER;
        LENGTH OF IBM-2314 IS 200
          CYLINDER;
```

In the above description, an IBM-2314 has been described as containing a basic storage cell of an 8 bit byte. The words BYTE, RECORD, TRACK, etc., have no special meaning in the language, but were chosen to make the description more readable. RECORD is defined as comprised of a minimum of 1 BYTE and a maximum of 7294 BYTE. TRACK is composed of RECORD, subject to two length constraints, namely a maximum of 71 RECORD and 7294 BYTE. This dual constraint is needed since the unit RECORD has a variable length, and therefore specifying only the number of RECORD per TRACK is not sufficient.

The remaining elements of the description should be obvious from the above description. The primary purpose of this information is to drive the device dependent accessor, which is described in a later section.

In the second part of the device description language, the addressing structure of the device is defined.

```
ADDRESS ENCODING IS DISK-ADDR;
    WIDTH IS 40 BITS;
    REPRESENTATION IS HEX;
ADDRESS FIELDS ARE CYLINDER-NUM,
  TRACK-NUM, RECORD-NUM;
CYLINDER-NUM FIELD IS 16 BITS;
    POSITION IN DISK-ADDR IS 0 THRU 15;
    RANGE IS 0 THRU 199;
    REPRESENTS CYLINDER WITHIN IBM-2314;
TRACK-NUM FIELD IS 16 BITS;
    POSITION IN DISK-ADDR IS 16 THRU 31;
    RANGE IS 0 THRU 19;
    REPRESENTS TRACK WITHIN CYLINDER;
RECORD-NUM FIELD IS 8 BITS;
    POSITION IN DISK-ADDR IS 32 THRU 39;
    RANGE IS 1 THRU 71;
    REPRESENTS RECORD WITHIN TRACK;
```

Here a 40 bit disk address named DISK-ADDR is described. An address is divided into fields, and the structure of each field, and what it represents, is specified. A permissible range of values for each field is given in the RANGE statement. Where the particular field is located within the address, is specified in the POSITION statement. The REPRESENTS statement relates the address field to the device hierarchy.

The accessing language

It is envisioned that a single specification of a device would be sufficient for any access method implemented on that device. The above description of the IBM-2314 could support many very different access methods for files on such a device.

The accessing language, on the other hand, describes how data on a device is to be accessed. This is done primarily by defining accessing primitives, which would then be invoked by a user to access the data. Once again, there are no predefined primitives. Any primitives which are needed must be defined in the accessing language in terms of access paths to a device.

Elements of the accessing language

Once again, this language will be presented in terms of an example. The example presented here borders on the trivial, but hopefully will at least give a flavor of the language, and how it is used to describe an accessing technique.

The access method presented here is for a sequential read of an existing file on the previously defined IBM-2314. In this example only the most basic functions are presented in order to minimize the complexity of the example.

```
ACCESS METHOD SEQUENTIAL-READ;
    DEVICE IS IBM-2314;
    STATUS INFORMATION IS CURRENT-
      ADDRESS;
        FORMAT CURRENT-ADDRESS IS
          DISK-ADDR;
    SPECIAL CONDITIONS ARE END-OF-
      TRACK, END-OF-CYLINDER, END-OF-
      FILE;
        CONDITION END-OF-CYLINDER;
            RECOGNITION ADDRESS FIELD
              TRACK-NUM IN CURRENT-
              ADDRESS GREATER THAN 19
            ACTION SET ADDRESS FIELD
              TRACK-NUM IN CURRENT-
              ADDRESS TO 0;
              INCREMENT ADDRESS FIELD
              CYLINDER-NUM IN CURRENT-
              ADDRESS BY 1;
```

```
CONDITION END-OF-TRACK;
    RECOGNITION HARDWARE;
    ACTION SET ADDRESS FIELD
      RECORD-NUM IN CURRENT-
      ADDRESS TO 1;
      INCREMENT ADDRESS FIELD
      TRACK-NUM IN CURRENT-
      ADDRESS BY 1;
CONDITION END-OF-FILE;
    RECOGNITION HARDWARE;
    ACTION RETURN ENDFILE
      STATUS;
```

In this first section of the language the format of certain status information maintained by the accessing algorithms is described. In this case status information named CURRENT-ADDRESS is being kept. The format of this status information is the same as the previously defined DISK-ADDR.

The remainder of this section defines certain exceptional conditions that can occur during the processing of the data. The criteria for the recognition of the special condition are specified, as well as the action that is to be taken when the special condition occurs. In some cases the recognition of the special condition is assumed to be performed by the hardware, while in other cases the accessing algorithms themselves must recognize the special conditions.

The final part of the accessing language defines the accessing primitives, and the semantics associated with each. Here only very simple semantics have been specified.

```
ACCESS PRIMITIVES ARE OPEN, READ,
  BACKSPACE, SKIP;
    ACCESS PRIMITIVE OPEN;
        PARAMETERS ARE FILE-ADDRESS;
            FORMAT FILE-ADDRESS IS DISK-
                ADDR;
        PROCEDURE SET CURRENT-ADDRESS
          TO FILE-ADDRESS;
    ACCESS PRIMITIVE READ;
        PARAMETERS ARE BUFFER-ADDRESS;
            FORMAT BUFFER-ADDRESS IS
              PRIMARY MEMORY ADDRESS;
        LOGICAL ACCESS UNIT IS RECORD;
        PROCEDURE TRANSFER FROM
          CURRENT-ADDRESS INTO BUFFER-
                ADDRESS;
            INCREMENT ADDRESS FIELD
              RECORD-NUM IN CURRENT-
              ADDRESS BY 1;
    ACCESS PRIMITIVE BACKSPACE;
        PROCEDURE DECREMENT ADDRESS
          FIELD RECORD-NUM IN CURRENT-
          ADDRESS BY 1;
    ACCESS PRIMITIVE SKIP;
        PROCEDURE INCREMENT ADDRESS
          FIELD RECORD-NUM IN CURRENT-
          ADDRESS BY 1;
```

In this simple implementation of a sequential access method on disk, a very basic OPEN function is specified. Here it is assumed that the accessing program provides the starting disk address for the file. In a more realistic implementation, the name of a file would be provided, and the specification for OPEN would cause a search of a table of contents on the disk volume for the starting address of the file. Needless to say, this would cause a great deal more complexity in the specification of OPEN.

The only primitive here which actually causes the transfer of information is READ. In the description of READ the unit of information transferred between the accessing algorithms is named in the LOGICAL ACCESS UNIT phrase. In this simple case it is assumed that a complete physical disk RECORD is passed to the accessing program. In more realistic descriptions, this simple correspondence between physical units of transfer and logical transfer units would obviously not hold. The procedure for mapping physical transfer units to logical transfer units would also have to be specified in this case.

The access primitive SKIP and BACKSPACE merely update the status information CURRENT-ADDRESS. Thus, on future invocations of READ the order in which records are returned is altered.

GENERALIZED ACCESSING ALGORITHMS

In order to assure the generality of the data access method being developed, the algorithms used in the method must be invariant, regardless of the data to be accessed. In other words, they must not depend on particular characteristics of the data. Although the development of such totally data independent, but practical, algorithms seems improbable, several approaches can be taken toward that direction.

The approach taken here is to identify a set of primitive operations of which algorithms used in various data access methods are comprised. These operations are primitive in the sense that the semantics of each operation must be unambiguous and simple, and parameters to direct each operation must be limited in number and well defined. Then by reducing the process of data accessing into a series of these primitive operations, and by varying parameter values for these operations, the algorithms which can be used for accessing various classes of data are obtained. In the model, these parameters are specified in the form of a high level language as discussed in a previous section.

As mentioned previously, it is not foreseen that a single set of algorithms will be sufficient to cover all cases of stored-data. However, hopefully pursuit of this approach will result in a limited number of sets of algorithms (or a single set of algorithms with multiple entries, if preferred) which cannot only be applied to any existing data access methods, but also easily be extended to new data access methods. Since the development of such algorithms is still in its infancy, presented here are functional descriptions of components of generalized accessing algorithms.

Components of generalized accessing algorithms

In discussing accessing algorithms, it is important to recognize two types of accessing units. One is a unit to which the data base (or operating) system provides a means of addressability. We term this a *physical access unit.* The other is a unit of information which is subject to access requests (i.e., a basic communication unit between the user and the system). This we term a *logical access unit.* This dichotomy is the underlying concept of the model.

As depicted in Figure 2, generalized accessing algorithms consist of three components: a Controller, a Device Dependent Accessor, and a Device Independent Accessor. The functional descriptions of these components will not be particularly novel in the sense that they can readily be seen in existing specialized access methods. However, they are different from those in conventional access methods to the extent that functions of each component are totally driven by the explicit descriptions of access methods.

Controller

The Controller is the part of the system which directs the entire operation of data accessing. Besides the normal function of coordinating linkages between various components of the system, the Controller has three additional functions to carry out: parsing of access requests, selection of an access path, and determination of access request fulfillment.

The Controller determines the validity of an access request expressed in a data manipulation language. Legitimacy of the request may include access security considerations. This process is performed by consulting structure mapping description tables which are obtained from the accessing description of the language. Once the validity of the access request is established, the request is transformed into a more convenient form for further processing. The complexity of this process depends on the allowable types of requests in the system.

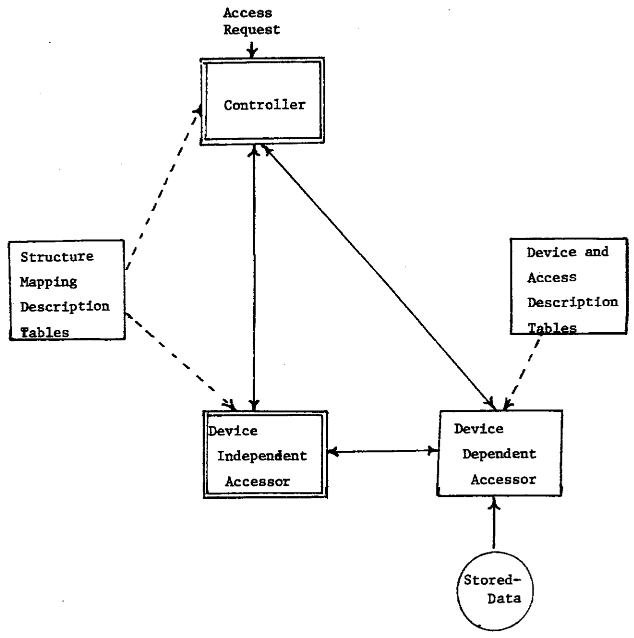

Access Request
Controller
Structure Mapping Description Tables
Device and Access Description Tables
Device Independent Accessor
Device Dependent Accessor
Stored-Data

Figure 2—Components of generalized accessing algorithms

Given the parsed form of the access request, the Controller then selects an appropriate (logical) access path to fulfill the request. It should be noted that there may exist more than one access path which can be qualified to satisfy the request. Therefore, the process of access path selection includes the determination of all access paths qualified, followed by the selection of the "best" one among them. This selection process is also driven by structure mapping description tables.

When a logical access unit is identified by the Device Independent Accessor, the Controller determines if it satisfies the access request. If so, the control is returned to the user. If not, the Controller requests the Device Independent Accessor to provide the "next" logical access unit on the access path selected. This request may, in turn, invoke the Device Dependent Accessor.

Device dependent accessor

Stored-data is an organized collection of physical access units. Its organization usually depends strongly upon the characteristics of a device on which the data is located. It is highly desirable that components of the accessing algorithms function as independently of these salient characteristics of devices and file organizations as possible. The function of this component is to remove such device and organization dependent characteristics from stored-data, so that the Device Independent Accessor can function independent of a particular device used and the addressing mechanism employed in the stored-data.

By examining the way that the access path selected by the Controller is encoded on the device, the Device Dependent Accessor transmits a physical access unit from the secondary storage into the main memory. This process is driven by a set of tables, called device and access description tables, whose contents are derived primarily from the device description. These tables are rather independent of the tables which drive the Device Independent Accessor. Thus, by driving the two components of the model using a set of independent tables, the model provides a very powerful and flexible means of accessing the data. By simply changing tables to be used by each component, the model permits the user to access the data, with different access and mapping strategies, which may reside on various storage devices.

Another function which must be carried out by this component is a problem resulting from the possible differences in elementary data representations between the system which created the data and the system under which the algorithms operate. This problem is a very common one when the accessing of foreign data files is considered. These architectural differences can also be resolved in the same technical approach (i.e. the description driven approach). However, it is felt that inclusion of problems arising from architectural differences is beyond the scope of our immediate research.

Device independent accessor

Given a physical access unit in a buffer, this component performs two major functions. The first function is to extract and identify a logical access unit from the physical access unit. Here the term "extract" is used to mean to separate one unit from another and the term "identify" to mean to recognize the name of the access unit. It should be noted that the order between the extraction and the identification of a unit is not definite. For example, the unit may first be extracted and then identified. On the other hand, the identification of the unit may be necessary to extract the unit.

The boundaries of physical access units may or may not correspond to the boundaries of logical access units within a particular data base architecture. In other words, the relationships between these two types of access units are in general $m:n$. Therefore, in order to extract a logical access unit, multiple physical access units may be required. This is accomplished by invoking the Device Dependent Accessor repeatedly.

The second function is, given a logical access unit as a result of the first function, to decompose it into its constituents. This decomposition process is carried out by the use of storage templates constructed from structure and mapping description tables. A storage template is a collection of named elements which schematically represent a logical access unit. One template is created for each type of logical access unit. It should be noted that a complete storage template for certain logical access units may not be constructed solely from the information in structure and mapping description tables. The construction of such a template may have to be deferred until the decomposition process of the access unit is initiated. In other words, some storage templates may be constructed dynamically.

CONCLUSIONS

It has been impossible in this paper to discuss in detail many of the really interesting questions one confronts when discussing access methods. Out of necessary space limitations, only the basic components of the model for generalized data access have been presented.

The question of generalized data access is by no means a solved one. The model presented here is intended as a research tool and not a production model. In the current languages the level of procedurality, particularly in describing access primitives, is much higher than desired. A great deal of emphasis in the design of the languages was placed on the factoring of information common to multiple accessing methods in common places, so that it would not have to be repeated. However, more work must be done in this area.

It is also felt that there are too many "pre-defined" keywords in the languages, which limit the generality of the model. Lower level descriptions of these current keywords are needed.

However, the current model gives us an important base to build upon. An implementation of the current model is planned, and it is expected that this will give us greater insight into additional problems. Even with the basic current model we now have the ability to model many different access methods in a common language. This alone has given us valuable insight into the nature of access methods, as well as having shown us weaknesses in the current model.

REFERENCES

1. Data Translation Project, *Functional Design Requirements for a Prototype Data Translator*, Technical Report, Ann Arbor, The University of Michigan, 1972.
2. ———, *Program Logic Manual for the University of Michigan Prototype Data Translator*, Technical Report, Ann Arbor, The University of Michigan, 1973.
3. Fry, J. P., R. L. Frank, E. A. Hershey III, "A Developmental Model for Data Translation," *Proceedings of the ACM SIGFIDET Conference on Data Description, Access and Control*, November 1972.
4. IBM Corporation, *OS/360 Data Management Services*, form #GC26-3746.
5. ———, OS/360 *Data Management for Systems Programmers*, form #GC28-6550.
6. ———, *Information Management System/360, version 2, General Information Manual*, form #GH20-0765-4.
7. ———, *2314 Direct Access Storage Facility and 2844 Auxiliary Storage Control*, form #A26-3599.
8. Senko, M. E. et al., *A Data Independent Architecture Model 1: Four Levels of Description from Logical Structure to Physical Search Strategies*, Technical Report RJ 982, San Jose: IBM, February 1972.
9. Severance, D. G., *Some Generalized Modeling Structures for Use in Design of File Organizations*, Ph.D. Dissertation, The University of Michigan, 1972.
10. Smith, D., *An Approach to Data Description and Conversion*, Ph.D. Dissertation, The University of Pennsylvania, 1971.
11. Storage Structure Definition Language Task Group, *Design Objectives for a Storage Structure Definition Language*, 1970.
12. Stored Data Definition and Translation Task Group, "An Approach to Stored Data Definition and Translation," *Proceedings of the ACM SIGFIDET Conference on Data Description, Access and Control*, November 1972.
13. Taylor, R. W., *Generalized Data Base Management System Data Structures and their Mapping to Physical Storage*, Ph.D. Dissertation, The University of Michigan, 1971.

A data base management problem specification model

by GERARD T. CAPRARO
Rome Air Development Center
Griffiss AFB, New York
and
P. BRUCE BERRA
Syracuse University
Syracuse, New York

INTRODUCTION

The data base management systems (DBMS) area has been growing at an accelerating pace over the past few years. There have been and are DBMS studies being conducted at many different government, industrial and university establishments. The work however, is primarily concerned with the software and hardware problems of building DBMS with unique capabilities. However, large scale research is not being conducted in the area of determining the interface between the DBMS and the data base problems they were created to solve. Each data base problem is as unique as the users of DBMS.

The data base problem must somehow be solved in the space of DBMS. Defining the data base problem in such a way that one can develop the solution in the DBMS space is no easy task. Presently, the user cannot define the data problem to the DBMS community in a language in which both can communicate effectively. This communication problem can result in a data base user obtaining DBMS that: (a) cannot handle the data problem efficiently; (b) will not handle the data problem efficiently in the future as the user's needs change; or (c) is much more powerful than the present and future needs require.

Some authors have addressed this problem. Dodd[7] and D'Imperio[6] consider some of the basic problems. Dodd points out that it was the user's needs that caused the existence of different data organizations or structures in DBMS. D'Imperio states that in order to take advantage of different data organizations, the system designer must have a thorough understanding of the data. Assuming that Dodd and D'Imperio are correct, it can be concluded that, if data are going to be placed efficiently in a computer, the data and their usage should be thoroughly understood before a data structure is chosen.

The philosophy of knowing the data and their usage thoroughly before choosing a data structure is fine in concept but in reality the people who are designing the system are not necessarily the people who are going to use it. Benjamin[1] looks at information system development from 1950 to the present and the communications that existed between the users and the designers of these systems. Between 1950 and 1964 typical information systems processed billing, payroll and stockholder records in a job shop environment as single functions. When the 3rd generation computer came on the scene (1964-1968), the number of single function systems had increased enormously and they were sharing parts of a now larger data base. At the same time, there was an increase in hardware and software capability which gave the system designers the opportunity to coordinate the single function systems into a more comprehensive system.

The important factor that Benjamin infers, but does not state, is that between 1950 to 1964, under a single function system, the problem-solver or user couldn't convey his/her total needs to the system designer or programmer such that the problem could be implemented on the computer in an efficient way. When these single functions were coordinated into a comprehensive system, the communications between the user and the programmer was still degraded. Benjamin makes this inference when he claims that the users' jobs relating to general decision and control were limited and that strategic management systems jobs for planning and modeling were deficient; "... in that their data requirements were not wholly consistent with that of the operational systems used by the rest of the organization." When Benjamin gives advice to system developers for "Mastering the Organization" he states that: "... the system developer must learn to understand the organization. Having learned how to write efficient code and how to integrate a number of small components into an effective and reliable system, he is free to tackle systems that encompass functional areas of the organization at different levels. ..." Benjamin goes on to state that one of the four things the system developer must do to master the organization is: "... to learn how to specify hierarchically large systems. Rapid progress will depend in part upon the development of specification languages that will allow for the complex definitions required. ..."

Other authors have expressed the need and impact of a user specification language. Merten and Teichroew[9] state in their conclusion: "... the design of problem statement languages and the design and construction of problem statement analyzers are formidable research and development tasks. In some sense the design task is similar to the design of standard programming languages and the design and construction of compilers and other language processors. However, the task appears more formidable when one considers that these languages will be used by non-computer personnel and are producing output which must be analyzed by these people. . . ."

The creation of problem specification languages for computer-based information systems is being attempted today. Teichroew[10] provides a survey of these languages and defines the specification for a "requirements statement language."

Therefore, the purpose of this paper is to present possible answers to the following questions:

a. What responsibilities in the designing or choosing of DBMS should be assigned to the data base user?
b. How can the user effectively interact with the DBMS designers to ensure that the data problem is treated properly once implemented on a digital computer?
c. How will the user and the system designer effectively communicate with each other once (a) and (b) have been determined?

A method by which this may be accomplished is to develop a model based on the fundamental factors involved in choosing or building DBMS for data base management problem. These fundamental factors should include a statement of the data problem, the computer hardware that is to be utilized for any particular data problem, and the computer software available on the computer system. The model should be such that the user's specification of the data problem can be utilized by the system designer to determine whether or not the data problem can be solved. There should be feedback to the user from the system designer in the same language in which the user specified the problem. This will enable the user to see how the specification of the problem has been changed due to system design problems and/or hardware or software limitations that may occur in the solution of the data problem.

The remainder of this paper presents a first attempt at a conceptual model that satisfies the above requirements. Additional work has been performed in the partial development of a user oriented problem specification language (PSL).[3]

DATA BASE MANAGEMENT SYSTEM PROBLEM SPECIFICATION MODEL

A purpose of this model, in the form of spaces and functions, is to try to provide a language for defining data base management problems and by so doing to find a solution based upon the problem definition. Even with the functions underived the model provides a language to help define the relationships of different work being performed in the data base management field.

Essentially, the use of the term spaces is an attempt to group entities in the data base management field so that those entities can be evaluated as to where and how they enhance the finding of solutions to data base management problems. Along with spaces are functions which are derived to describe the interrelations between the spaces. The grouping of these spaces and functions will, hopefully, help in describing the data base management system model and provide a language criteria for discussing data base management system design and evaluation.

The Data Base Management System Problem Specification Model (DBMPSM) can be described as shown in Figure 1.

Let the input space, I, be made up of sets whose elements are a subset of the attributes of the total data base. Each set shall describe either a physical form in which data may enter the data handling system or any data maintenance which by being performed is equivalent to a physical form of input data. Sets of the space I could be magnetic tapes containing digital test data, personnel forms, questionnaires, etc.

Let the output space, O, be made up of all the questions (or jobs) that a user may wish to ask of the data base plus any maintenance that has to be performed on the data periodically within the data handling system. Elements or tasks contained in the space O can be defined as sets where each set could be updated personnel files, calculating statistics from stored data, performing queries on value limits of particular attributes, etc. These sets shall contain the attributes needed to perform the requirement defined by the sets that make up the space O.

P & V are the functions that will develop a problem specification language (PSL). V(O) is a function that describes the requirements on the space PS due to the space O. P(I) is a function that describes the requirements on the space PS due to the space I. The union of these two functions will define the problem specification language.

The space problem specification, PS, is defined by the data base and the user's requirements through the utilization of the PSL.

The space PS will bridge the communications gap between the user and the computer systems designer. It will provide a specification of the data problem in a language that both the user and computer software people can utilize to maximize the probability of determining the proper solution to the first data problem question of "does the user need to enter his/her data onto a digital computer?" If the answer defined by the interpretation of PS is no, then PS should be used to help define what is needed. If the answer is yes, then the functions Q_1 and Q_2 can be utilized to help determine the logical file structures needed.

The logical file structure (LFS) space contains all current

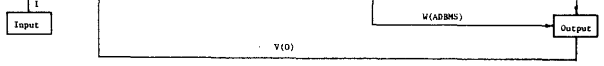


Figure 1—System problem specification model (DBMSPSM)

logical file structures available from the available data base management system (ADBMS) space. Examples of these file structures could be random files, random serial files, tree file structures, chain file structures, etc.

The function Q_1 is defined as a function that will provide the capability of defining the logical file structure Q_1(PS) specified by the problem specification. If Q_1(PS) is not available within the space LFS, then one of two alternatives are available.

(1) Q_1(PS) can be used as the logical file structure and in conjunction with the space PS a solution of the data base management problem can be solved by building a tailored solution space.
(2) The function Q_2 can be utilized. The function Q_2 provides the capability of defining which logical file structure in LFS defined by Q_1(PS) and will also define the changes to the space PS that have to be accepted if the logical file structures in the space LFS are accepted. This is vitally important to the user for he/she can determine by the changes made to space PS the impact that will occur because the specification of the problem has been changed.

If, however, Q_1(PS) does define a logical file or files contained within the space LFS, then a Data Base Management System and/or computer physical storage structure can be developed.

The space, ADBMS, contains all the Data Base Management Systems available to the user. For each DBMS in ADBMS all of its pertinent factors are contained and described, i.e., physical data storage techniques, query language, special software and hardware capabilities, logical file structures, etc.

The function R_1 is defined as a function that has the capability of defining the physical file storage structure or structures R_1(LFS) dictated by the logical file structures contained in the space LFS or derived by Q_1(PS). If R_1(LFS) is not contained in the space ADBMS, then one of two alternatives are available.

(1) R_1(LFS) can be used as the computer physical file storage structure and in conjunction with the space PS, a solution of the data base management problem can be solved by a tailored solution space.
(2) The function R_2 can be utilized. The function R_2 will provide the capability of defining which physical file storage structure(s) in ADBMS is closest defined by R_1(LFS) and will also define the changes to the logical file structures in LFS that will have to be accepted. This again, as stated for alternative (2) in utilizing Q_2(LFS), provides the user a feedback capability through Q_2(LFS) to evaluate the impact that will occur because the specification of his/her problem has been changed.

If, however, R_1(LFS) does define a physical file storage structure contained within the space ADBMS, then the DBMS associated with this structure can be chosen. If, however, there are more than one physical file storage structures defined by R_1(LFS) and no DBMS in the space ADBMS contains all of the physical file storage structures, then the user is faced with the same two alternatives as described above.

The function W maps into those elements of the output space, O, which can be provided for directly, because of the DBMS chosen in ADBMS. Certain DBMS' have inherent capabilities that can provide all or some of the needs that are defined by the output space. These capabilities could be such things as queries on files due to specific values or ranges of values, file editing, report production capabilities, etc.

The function S is that function which in conjunction with W(ADBMS) defines the output space, $S(ADBMS) \cup W(ADBMS) = O$.

The standard software program space, SSP, is that space containing software programs resident on the computer system chosen. If $S(ADBMS) \subseteq SSP$, then $SSP \cup W(ADBMS) = 0$. If, however, $S(ADBMS) \not\subseteq SSP$, then external software routines must be written.

These external routines are defined by function T; where $T(SSP) = S(ADBMS) - [S(ADBMS) \cap SSP]$. They can be programs coded, for example, in a standard compiler language (i.e., FORTRAN IV, COBOL, etc.) in conjunction with a special language of the DBMS (query language).

CONCLUSIONS

This research effort was undertaken to try to develop a feasible model in which the user of a data base could effectively interface with the design and/or choosing of a data base management system. Over the past few years the data user has been slowly taken out of the sequence of events that lead to the solution of his/her data problem on a digital computer. The conceptual model developed, DBMSPSM, is a first attempt to put the data base user as the driving force in finding a solution to his/her description of the problem.

To develop all of the functions and spaces that make up the DBMSPSM appears to be a very large task. The first portion that must be performed in this task is to develop a problem specification language (PSL). This must be accomplished first since it is the basis of the model and all feedback to the user must be described in the PSL. Once this PSL is completed it should be tested on an existing data problem to determine its feasibility and ease of implementation in trying to specify a data problem (derive the problem specification (PS) space). If this proves successful then the functions (Q_1 and Q_2) necessary to map the PS space into the logical file, structure (LFS) space need to be derived. This procedure of derivation and testing should continue until the DBMSPSM is completed by having the entire model implemented on a digital computer.

REFERENCES

1. Benjamin, R., "A Generational Perspective of Information System Development," *Communications of the ACM*, Vol. 15, No. 7, July 1972, pp. 640-643.

2. Bloom, B., "Some Techniques & Tradeoffs Affecting Large Data Base Retrieval Times," *Proc. ACM 24th Nat. Conf. 1969*, pp. 83-95.
3. Capraro, G., *A Data Management System Problem Specification Model*, RADC-TR-73-193, June 1973.
4. Chapin, N., "A Deeper Look at Data," *Proc. 23rd ACM. NAT. Conf. 1968*, pp. 631-638.
5. Chapin, N., "Common File Organization Techniques Compared," *FJCC*, 1969, pp. 413-422.
6. D'Imperio, M., "Data Structures and their Representation in Storage," *Annual Review in Automatic Programming*, Vol. 5, pp. 1–75.
7. Dodd, G., "Elements of Data Management Systems," *Computing Surveys, ACM*, June 1969, pp. 117-133.
8. Mealy, G., "Another Look at Data, *FJCC*," 1967, pp. 525-534.
9. Merten, A. and D. Teichroew, "The Impact of Problem Statement on Evaluating and Improving Software Performance," *FJCC*, 1972, pp. 849-857.
10. Teichroew, D., "A Survey of Languages for Stating Requirements for Computer-Based Information Systems," *FJCC*, 1972, pp. 1203-1224.

Integrating data base management into operating systems—An access method approach

by ALBERTO CEZAR DE SOUZA MOREIRA

The Light and Power Company
São Paulo, Brazil

and

CLAUDIO PINHEIRO and LUIZ FERNANDO D'ELIA

UNIVAC Brazil
São Paulo, Brazil

INTRODUCTION

It appears that the final word about Data Management and Data Base Management has not yet been said. Different authors stress different points in the rather sparse bibliography on the subject. Different program products and packages are oriented toward completely distinct approaches, diverging progressively from a standardization rather than converging to it.

Therefore the authors felt inclined to attempt a small contribution in this field by establishing their own approach toward the problem of implementing a data base software that behaves according to those concepts. The bases of the adopted approach are: (a) to integrate closely the data base management software with the operating system as an access method, (b) to implement user interfaces that are similar to the standard data management access methods, and (c) to concentrate in flexible data structures and low core usage.

This paper intends to go beyond the usual problems of data independence, transition from non-database methods, standardization and others. It intends to present an understructure over which to build the solutions for the aforementioned problems. It describes the authors' implementation of the proposed approach and discusses some of the important concepts involved. Data structures, retrieval logic and user interface are examined and their relationships with the overall approach is enlightened.

GENERAL APPROACH—AN ACCESS METHOD

The Data Management function in a computer system is the interface between the user programs and those parts of the operating system that deal more directly with the Input/Output devices and their hardware.

Data Management provides centralized services to the active users in the multiprogramming mix. It interchanges user and files, performing housekeeping and maintenance tasks and creating an adequate environment to file access and query. It manages computer resources granting access to Input/Output devices and sharing them between the users and at the same time securing data against errors and accidents.

The authors cannot think of Data Management as a program package or product. It is an integrated part of the operating system, interfacing with it at a very low level. User programs use Data Management as a normal communication path with the operating system, forwarding requests and receiving services from the facilities implemented. To implement those basic access facilities as a set of user programs would not give any advantage over the integrated approach. It would certainly add an extra level of system complexity and overhead, for the interface between user programs and files would be established at user program level (which in turn would have to interface with the user files through some basic access routines) rather than at very basic physical Input/Output level.

A data base oriented Data Management System should be no exception. From the software point of view a data base is a set of user data files in which structural relationships between the existing data elements have been implemented. What is desired here is to have the capability of accessing data fields, segments, records or other structures by name, relieving user programmers from the physical aspects of this usage. This is not different from the aims of traditional Data Management; only the requirements placed on the flexibility and power of the implemented data structures are much more stressing.

Data must be accessed in multi-sequential, random and hierarchical orders; programs must be as independent of physical data layouts as possible; multiple users and multi-file data bases must be accommodated. And all this must be done in such a way that user programs are kept completely unaware of the physical side of the hardware and software.

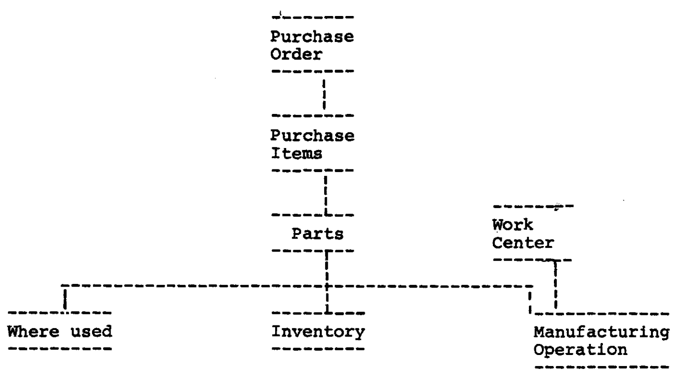


Figure 1—Hierarchical relationships

This is the approach that the authors have chosen to implement DBAM. A Data Base Access Method—this is the essence of its design. It aims to put data base resources at user's hand, integrating him with the operating system at the lowest possible level. The standard user interface allows the user to communicate his needs to the operating system through DBAM is an integrated part of the operating system, presenting a common interface together with all other access methods and preparing an easy building of the high level language standard interfaces by simple and straightforward compiler extensions.

Another important point considered by the authors was to divorce data base management from data communications. Handling remote terminals and concentrators, queuing and dequeuing messages, must be the object of another access method. Some interactions will exist in the sense that certain storage techniques when handling telecommunications may need interfaces between the Data Transmission Access Method and one of the file management access methods available, but these are just interrelations between the various components of the operating system rather than user programs exchanging data. From the standpoint of a user who wants to implement an on-line applications program, it is important that the operating system has the capability of both dealing efficiently with telecommunications Input/Output and allowing the implementation of a minimum access time file structure oriented to the particular application. This does not imply necessarily a close integration between the data communications and the file retrieval capabilities, but a simultaneous integration of both the data communications and the data management with the operating system in such a way to permit the user to express his needs in both fields.

The data base access method is therefore an operating system component available to service his users in a shared way. Its design must express a tendency toward excellency and high service level; the usual requisites of Data Management functions must be met. Sequential and random retrieval, associative and hierarchical structures, multilist and inverted list processing, low core usage, fast response times and efficient management of system resources must be complemented by management-oriented features like easy transition, data independence and data security.

This approach reflects the belief that the interface between the operating system and a data management function can be advantageously placed at the access method level. It brings easy transition from non-database programs through the common interface presented to the operating system by the access method. It connotes a powerful centralized software that can be in the future the standard logical I/O interface for the whole operating system together with its compilers, service processors and utility routines.

DBAM was designed to operate in a wide range of computers and operating systems. It can be run with small modifications in virtually any byte oriented machine, from very small to very large configurations. Users have available a standard and general software able to work in a variety of environments.

DATA STRUCTURES AND ADMINISTRATION

General concepts

An effective data management access method must be able to handle a variety of data structures and organization methods. Multiple hierarchical and associative relationships, sequencing by multiple sort fields and flexible list manipulation schemes must be available.

Data elements

Three basic data elements are recognized by DBAM: fields, segments and records. Fields are the smaller units of data with a logical meaning. Segments are sets of fields stored together as a whole and bearing a logical relationship. Records are collections of hierarchically related segments.

A user program to control loading in distribution transformers may have, for example, two segments of data: the transformer segment with transformer data and the consumer segments with all data of those consumers electrically connected to that specific transformer. Individual data such as transformer maximum load or consumer connected load are some of the fields of our segments. A particular transformer segment with its associated consumers is the transformer record. Furthermore, we might have a feeder record composed by a feeder segment and all segments related to transformers connected to that feeder.

Hierarchical relationships

The subordination of a segment to another is what is basically understood as a Hierarchical relationship. The subordinate segment is called "Parent," and the subordinated is called "Child" or "son." A given parent can subordinate many sons under the same specific relationship, or it can subordinate several hierarchical relationships with many sons each one. Also a son can be subordinated by several parents, one for each hierarchical relationship of which that son takes part. A dependent segment may for its turn subordinate several hierarchical relationships of its own, allowing a multi-level multi-parent multi-child tree of hierarchical relationships.

As an example, consider a sample manufacturing control system. In Figure 1 it is shown the relationship between a part and its manufacturing process by means of the hierarchical relationship between a Parts segment and an Operation segment. Also, an inventory segment is depicted as the necessary subordination for an inventory control application and a Where-used segment to help in the bill of materials preparation. For a scheduling operation, work center segments relate the manufacturing operations to the work centers where they are performed. Also, for purchasing, purchase order and purchase items segments are necessary. The structure of Figure 1 is therefore satisfactory to a range of different application programs.

Associative relationships

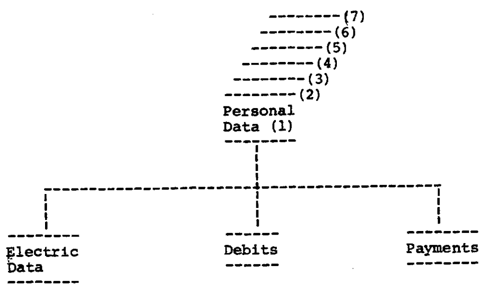
Many applications need that association between data segments or records based on the equality of a certain field. Furthermore, it may be necessary to access sequentially all segments having some common feature or satisfying a set of common criteria such as fixed field values, segment type, keys and others. This kind of segment relationship is called an Associative relationship.

It is possible to include a data segment simultaneously in a number of associative relationships. The segment can also participate of hierarchical relationships of its own.

As an example of simultaneous hierarchical and associative relationships, consider the case of Figure 2.

In the electric billing application depicted, a customer has up to four different segment types. The Electric Data, Debits and Payments segments are hierarchically subordinated to the Customer Personal Data segment. This subordination will allow the applications programs of the Billing System to selectively process the customer records and to worry only with data. However, there will be a need to process customers according to certain particular characteristics such as type (commercial, industrial, residential), city district, connected load range, etc. It is possible then to associate all Customer-Personal Data segments according to those criteria, defining an associative relationship for each valid common characteristic. Notice that by defining the relationships at the segment level it is possible to associate the customer data record as a whole, for all other data segments are hierarchically subordinated to it.

Associative relationships have a number of optimizing features available. Searches are always performed using the shortest path available. Sequence fields are checked before the search starts and proper location to begin retrieval is found, saving many mass storage accesses. Special provisions have been made for inverted list (that is, list length = 1) search and processing.



```
Personal Data: (1),(4),(5),(7) = residential customers
                                 associative relationship

             : (1),(2),(3),(7) = associative relationship
                                 by district.
```

Figure 2—Simultaneous hierarchical and associative relationships

Data base organization

The user data base is a set of files where data is stored in a blocked direct access format. Block sizes are fixed and specified by the user for each one of the data base files. Access is done at the physical input/output level. Segments are fit into the file blocks and packed together so that different segments can be specified.

The addressing technique uses two different location modes. It is possible to have both directly addressable and non-addressable segments. Independently of the characteristics of the hierarchical relationships defined for the segment, an addressable segment must have a key that goes through a scrambling process called 'Randomizing' to generate a calculated disc block address. Segments are then fit in that block if space is available. If no room is available in that block, a new trial is done based on user specifications. The maximum number of trials before a no-room or no-find condition is generated is also a user choice.

For each defined relationship a double 4 byte pointer is generated to point both to the next and to the previous segment in the list. This technique allows extensive file update without need of file reorganization and also permits the implementation of broken chain methods for list reconstruction.

It is also possible to increment the size of a file without reorganizing the data base. Any new file extent, up to a certain limit, will be recognized by the access method and bring an automatic adjustment to the new situation.

Data base control

Data base control is done by using a 'Schema' file that contains all pertinent information about data structures and characteristics. Segment, record and field descriptions, associative and hierarchical relationship control data, file descriptions, list headings and queries are stored. Program Information Blocks, that interface user programs with the data base, are also resident in the Schema file.

With the aid of file, data and relationship descriptions—DBAM access the data base to perform user writes and reads. Retrieval logic is controlled by the stored queries. Automatic search and retrieval operations are possible by

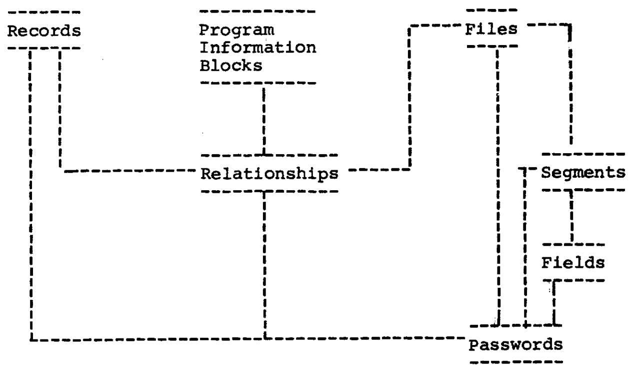


Figure 3—Hierarchical relationships in the schema file

the use of Program Information Blocks. These PIBs are treated as 'registrations' of user programs in the data base. They contain interface information, statistics and the data base path the program sees, so that an automatic retrieval operation following these relationships is possible.

The data in the Schema file is grouped in segments and stored in the same format used to maintain the user data base. Several hierarchical relationships are established between these segments. They help the user to control and audit his data base and his user programs. Data dictionaries, cross-reference listings involving programs, data and relationships, periodic structure audits and checkups, are facilitated. Figure 3 contains the main hierarchical relationships implemented in the Schema file.

ACCESSING THE DATA BASE

Retrieval logic

The retrieval logic is built in such a way that the user "sees" a straightforward structure of relationships connecting his data, orienting him to follow those relationships just like he follows the roads and streets that compose the way from his house to the office.

The user sees a two-dimensional grid formed by the associative and hierarchical relationships that were established at data base generation time. The data base is then a collection of segments linked by both associative and hierarchical dimensions.

Those dimensions are independent but not equivalent. Associative relationships are treated as global entities, relating segments and records in an overall manner. The associative capability is used to establish main paths within the data base and allow generalization. The hierarchical dimensions in a local entity, as a search or a sequential retrieval will always depend on the existence of a parent that restricts the operations to the local level.

Suppose (recall Figure 2) that we want to retrieve the debits of all residential customers of a given city-district.

The relationship that ties up all Customer segments of that district is an associative relationship. The tie between a Customer segment and a Debits segment is a hierarchical relationship.

When retrieving the debits, the user logic could be:

1. Global search on the associative relationship that ties all customers of that district and retrieve the next residential customer.
2. Local search on the hierarchical relationship that ties the Customers to the Debits in order to pass to the user the Debits segments.
3. Repeat steps 1 and 2 until list end.

Once the global search is started, the local search of item 2 does not disturb it. It is always possible to perform a search in one of the dimensions independently of the other.

The user may interrupt a sequential search and retrieve data randomly. He may or may not return to the original search. He may define a pre-established sequence of hierarchical relationships in a Program Information Block and do an automatic search based on that sequence. He may also do a retrieval based on segments rather than on relationships, and, in local searches, he may progress "bottom up" retrieving hierarchical parents rather than hierarchical dependent segments.

The user always has a "current position" in the data base. Also there will always be an active associative relationship and a current active hierarchical path. The DBAM retrieval module "remembers" its most recent hierarchical path and also keeps track of all path interchanges. The active paths represent "roads" he can choose at retrieval time and are signaled by the interchanges DBAM can remember.

As an example let us consider the case of Figure 4. The user might search all segments under the "*a*" associative relationships until he finds A_n. At this point, A_n is the current position and "*a*" is the active relationship.

User might then start a hierarchical search to find segment B_2 and then segment C. It is possible now to either go back to A_n and restart the associative search (that is still active) or retrieve segment E, or retrieve segment D_1 backing up to segment B_2. The path already travelled is remembered together with all interchange points like A_n and B_2.

If an associative relationship was defined for segments type D, the user could stop the hierarchical search and proceed through this new associative path which would then become active in place of the previous active relationship "*a*"; also all previous hierarchical paths would be lost because a new global search would be starting.

Qualifications

With qualifications the user can perform selective search according to specified parameters. A qualification is a restriction that then may be established in order to selectively retrieve certain segments or record.

The qualifications provide also the means to pass key values, relationship names and segment or record types to

DBAM. Among the entities that may be used to qualify a retrieval are:

a. a relationship name
b. a segment name
c. a record name
d. a Program Information Block name
e. a key value
f. a field value

Theoretically the user may have a request with no qualification. All items are either optional or defaulted.

Following are some sample qualifications.

a. To ask for a EMPLOYEE segment:
(*SEGMENT.EQ.EMPLOYEE)
b. To specify a key value:
(*KEY.EQ.950032)
c. To limit a field VALUE to be less than 45:
(VALUE.LT.45)
d. To include a Program Information Block in the qualification:
(*PIB.EQ.FT325V10)

The qualification can be associated by logical AND, OR or NAND (AND NOT) connectives. For example: the qualification

(*SEGMENT.EQ.EMPLOYEE).AND.(AGE.
LT.45).AND.(JOB.EQ.ENGINEER)

means that the user wants EMPLOYEE segments for employees less than 45 years old who are engineers.

The OR connective is used to connect two different qualifications and means that a list merge is to be done. The qualification

(*SEGMENT.EQ.EMPLOYEE).AND.(AGE.
GT.60).AND.(JOB.EQ.DIRECTOR). OR.
(*SEGMENT.EQ.EMPLOYEE).AND.
(FATHER.EQ.PRESIDENT)

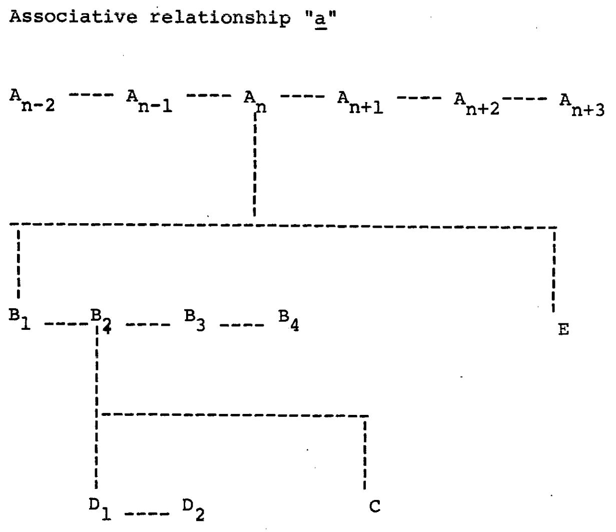
means that all segments that describe either directors older than 60 or employees who are sons of the president will be passed to the user.

A set of qualifications is called a QUERY. The user may store queries in the Schema file and retrieve them at run time. Therefore, the Schema also contains a "Query Library" where queries are stored as segments in the common DBAM format.

When handling a query, DBAM tries to make the better use of the data base structure. If a global search is made over an inverted list, list headings are merged or intersected. If two or more associative relationships are intersected, the search is made in the shorter path. If sequence data is specified, the sequence fields in the list headings are searched and the proper sublist is used as a starting point.

Writing

Writing on the data base is controlled by the Schema Control data entered at Data Base Generation time. The Schema contains descriptions of all relationships, sequence fields and sequencing options applicable to user data.



Associative relationship "a"
A_{n-2} ---- A_{n-1} ---- A_n ---- A_{n+1} ---- A_{n+2} ---- A_{n+3}
B_1 ---- B_2 ---- B_3 ---- B_4
E
D_1 ---- D_2
C

Figure 4—The DBAM retrieval logic

USER INTERFACE

Introduction

The user interface with Data Base Access Method involves a two-sided approach. Data Manipulation Language will be used by application programmers and systems analysts. Data Description Language will be a tool for a Data Base Administrator or other high level professional in the Operations area.

Data description

A major point in data description is to make language context and concepts as close as possible to the real data structure. It is easy to understand data structures as associations of records and fields by means of relationships, but the introduction of new words, structures and abstract concepts evolved from the computer science practice widens the gap between systems analysts and Data Management.

A simple language is the best approach. Data Description Language must be data and structure oriented. One record for each data base entity, that's all that is needed. Cross references, duplicate information or redundancies must be avoided. The authors believe that adapting data description to high level languages as COBOL or PL/I may make data maintenance cumbersome and complicated due to the natural inefficiency of the language context.

The approach used in DBAM is to have a command for each file, data element, relationship, program or query stored in the Schema file. Information is entered only once. No double checking or pairing of commands is needed. No extra training is needed but the general understanding of the implemented structures and concepts.

Data manipulation

There are some major objectives to be achieved by the design of an interface between a user and his Data Management software. First, a common interface in assembler language must be written so that all other software may communicate with Data Management at a basic level. From this level, all language processors may generate assembler language sequences to access Data Management from high level language user verbs and specifications.

This approach is widely used in non-database Data Management. It should be no different in data base oriented Data Management. A user programmer must be able to OPEN, CLOSE, GET and PUT just like in any other access method. There are several advantages in using this approach, like easy transition from non-database methods, fast programmer and systems analyst training and simplicity of data base manipulation.

As an example, let us consider the basic user interface in OS. The user program must build a Data Base Control Block for its data base which contains basic data base information and behaves just like a standard DCB. It is built using most of the standard OS keywords as MACRF, DSORG, DDNAME, EODAD, OPTCD and others. A job control language DD statement may also contain DBCB information.

An OPEN statement initializes processing, building up linkages to the access method and its buffers. SETL statement initializes associative and hierarchical searches using relationship names. GET and PUT are used to access user data in the data base or insert new data. Also a LOCK command may be used to update a data element, granting exclusive access until the operation ends.

CONCLUSIONS

In the present paper the authors intended to defend a number of points regarding Data Management and Data Bases and present a real implementation of a Data Base Access Method in which the concepts presented are implemented.

DBAM runs in 16K bytes under OS, demonstrating that high effectiveness and low core usage can be simultaneously attained. Also a wide range of data structures can be implemented, from hierarchical relationships to multilist and inverted list associative links.

Data Base management can be interfaced with the operating system at a very basic Input/Output level. It should be an access method rather than a program package. Its communication with the I/O devices should be done at physical I/O level. The adoption of this philosophy brings steady evolution from older systems by progressive and simple change of the access methods as well as keeping the original operating system philosophy as much as possible.

A Data Base Access Method can be a standard interface for all the components of the operating system. The whole non-removable mass storage can be thought as a data base within which Source Programs, Load Modules, Work Areas, Message Queues and other system entities are the active segments. The logic of the operating system can be implemented as data relationships between the segments, saving a lot of programming headaches. Operating efficiency and programming and systems analysis economies would be achieved just like any commercial oriented data base applications program. Core usage and running time would certainly decrease.

Data Base Management and Operating Systems integration may be the key for future developments in this area.

A prototype system for interactive data analysis

by GERALD LEVITT, DAVID H. STEWART and BEATRICE YORMARK

The Rand Corporation
Santa Monica, California

INTRODUCTION

The analysis of small and simple data collections is commonly accomplished through the application of "canned" statistical analysis programs. For larger more complex data collections, however, such programs often do not satisfy a researcher's needs. In these cases, the additional use of specially developed computer programs may be necessary. These programs frequently require modification and reformatting of data to meet their input requirements. These additional complexities are compounded when a researcher attempts to investigate alternative hypotheses or pursue hunches requiring further transformations or restructuring of the original data collection. Often, this process involves the services of a professional programmer making repeated program modifications and computer runs.

To study these difficulties, a prototype computer system called the Data Analysis System (DAS) was developed which aids researchers in accessing their data and assisting them in interactively applying a variety of standard analytic procedures in a unified and consistent manner. Through a comprehensive graphical terminal, researchers are able to: review data in tabular or graphical form, subset and restructure the data for hypothesis testing and formulation, and apply many standard statistical tests.

The DAS, although similar to other systems addressing a data analysis capability, differs from those systems in one or two basic respects: (1) the availability of a natural definitional language to manage and restructure data collections; and (2) the ability to easily and quickly form graphical presentations of data both in their collected and restructured forms.[1-4]

This paper presents both the basic design notions used to develop the system and a functional description of the facilities it provides.

DESIGN CONCEPTS

Data analysis concepts

The Data Analysis System (DAS) design goals have drawn heavily upon Tukey's characterization of the data analysis process. Five of these characteristics important in the design of the DAS are discussed below.[5]

1. *Summarization*

Summarization is viewed as the process of formal statistical description. It consists of using statistical models to test for hypothesized relationships. Applications of summarization techniques include the ability to use the residuals of summarization processes as data. An example of this in the DAS is the ability to display the results of a linear regression and a plot of its residual against explanatory variables.

2. *Exposure*

Tukey defines *Exposure* as the "... effective laying open of data to explore the unanticipated". This can be accomplished by using standard statistical methods on the data in a flexible way to elucidate new hypotheses and reveal possibly unknown relationships. It is the ingredient that has been missing in computer-assisted data analysis because it requires a level of informality in the use of techniques not normally considered within the domain of formal statistics.

3. *Iterative Nature*

The data analysis process is characterized by the repeated restructuring of data collections and multiple applications of summarization and exposure techniques. This process is intrinsically iterative—no step is clearly the last before it is taken. Human judgment is needed at almost every step. The analyst must be allowed to flexibly choose a model for summarization and apply it to any set of data. An important goal of DAS is to facilitate the interplay between exposure and summarization.

4. *Scaling of Data*

One of the goals of data analysis is the search for simplicity in the description or explanations of relationships between variables.

To permit the use of simpler analysis models there must be a facility for easy transformation of variables.

5. *Missing Data*

Often data used in the analysis process suffer from missing observations. Dealing with missing values, particularly in multivariate forms of analysis, can be a problem. The patterns of missing observations often vary from variable to variable. Data in this form can result in a loss of predictive strength due to the loss of numerous observations.

An important aspect of the DAS approach is to provide a means of easily identifying, manipulating and managing missing observations.

Information language

One of the important goals of the DAS effort was to produce a system that could be used naturally by an analyst. In order to represent the system to these users and to allow them to use it in a natural way, we formalized an information language for data analysis. This information language presents terms and constructs for dealing with the data and analysis process. Several of the important aspects of this information language are discussed below.

1. *World View*

The world view captured by the information language of the DAS sees data bases and operations on them as functions of the analysis process rather than as an information retrieval system providing file retrieval. The DAS language therefore allows the user to define, label and indicate how data groups are to be used in the analysis process without requiring the user to either understand or deal with the underlying files and data manipulation mechanisms.

2. *Nature of the Data*

A great deal of the data for analysis is collected and stored in discretely identifiable units called *cases*. Cases may be in part an artifice of the data collection process or in part a predetermined structure specified by the analyst.

3. *Attributes*

The data of a case can be conceptually divided into different categories called *attribute classes*. The term *varible* is often used to connote the same property as an attribute. An attribute class is defined by its name and associated set of values which may be: empty; consist of only one datum; or consist of a large class of related data. For example, in hospital data the attribute "patient name" would be associated with one datum, the attribute "patient temperatures" with series of data and the attribute "patient age" null if that datum was unobtainable. A particular case, in turn, is completely defined by the enumeration of its contribution to attribute class values.

4. *Sets*

Cases that are described by the same attribute classes may be collected to form a group called a *set*. Each case in a set is assigned a unique identifier used thereafter to refer to the case or a particular attribute value of a case. Conceptually, a set may be thought of as a matrix whose rows are labeled by case identifiers, and columns labeled by attribute names. Sets play a very important role in the process of data analysis. Many of the activities of data analysis specify, construct and evaluate sets.

User interface

Of primary concern in the design and implementation of the prototype system has been the interface between the user and the system. It is of utmost importance that the medium of communication be convenient and appropriate to the data analysis context, resulting in users feeling as if they were dealing directly with their problems.

Because the analysis environment requires facilities as broad in scope as data management, statistical analysis and graphing, the tools the analyst must use to carry out these functions can become exceedingly complex and require many steps or procedures to accomplish a single task. It is with a respect for the power of graphic presentation and its relationship to other analytic tasks that we have placed a primary emphasis on user control of the system through a graphic medium.

The methodology employed in creating the interface utilizes a display tube with a sensitive surface at which the user can point to invoke system responses. In addition to graphs and diagrams, the objects displayed on the screen include menus of functional options of the system, tutorials, and requests by the system for action from the user.

A general theme throughout this communication is that the user should be informed of what is expected of him, whether or not he has control, and, what the system is doing. Also incorporated in the user interface is an extensive subsystem of tutorials used to further clarify the user-system status. Used in another mode, tutorials are presented to explain the state of the system and the standard conventions of the hardware and data analysis.

The interface assists the user in selecting and using the functions of the system. In turn, it elicits the specific functions requested and re-receives control of the system when a particular function has terminated its activities. Since functional modules can elicit modules subordinate to them, another task of the physical user interface is context management. By context management is meant performing the housekeeping to determine how a function was entered and, as a result, determine the alternatives and actions necessary to return to a previous context, go to a new context, or return to the initial context. An important task of the user interface is to make this complexity transparent to the user.

FUNCTIONAL CHARACTERISTICS

The initial display context

The DAS provides its users with two basic classes of capabilities. The first class consists of operations which are executed on sets; they include the creation of sets, the deletion of sets and the display of set data and associated data summary statistics (e.g., max., min., avg.). The second class consists of a variety of statistical methods which can be applied to the data contained in sets; these include the histogram, scattergram, plot, barchart, crosstabulations (2-way) and stepwise linear regression.

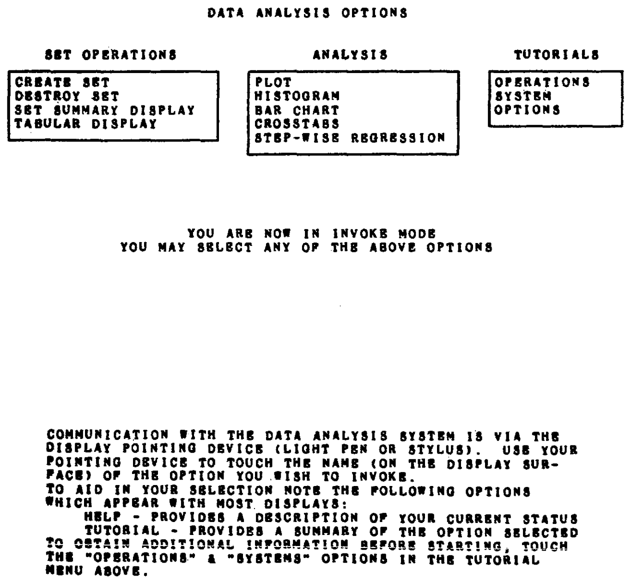
These capabilities are presented to the user for their selection as options in our initial display context. The options are organized in two display menus as pictured Figure 1. A third display menu provides the user with an added set of options which when selected will generate tutorial information about the system itself, its operation and each of its set manipulation and analysis capabilities.

Normally, a user invokes the desired capability by touching the appropriate option name in one of these menus; when this happens the display context immediately becomes that of the option selected. If the user first selects *OPTION* in the tutorial menu, however, the context does not change. Instead, the user may then point to an item in any one of the other menus to display tutorial information about it. The user may easily switch from the invoke mode to tutorial mode and back again by touching the appropriate sensitive areas.

This initial display context is presented when the user first logs on to the DAS and when he returns to it from some other context.

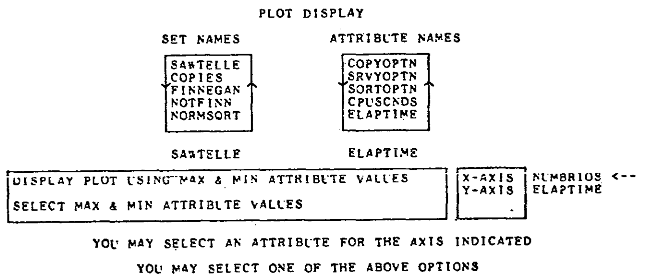
Selection of sets and attributes

All system analysis capabilities require the selection of a set and one or more of its attributes before that capability can be executed. The process involves two menus. The first is a menu of set names. This menu is displayed automatically whenever it is needed. On selecting a set from this menu (using the pointing device), a second menu containing the attributes of that set is then also displayed.



```
                         DATA ANALYSIS OPTIONS

    SET OPERATIONS                 ANALYSIS                  TUTORIALS
 CREATE SET                PLOT                          OPERATIONS
 DESTROY SET               HISTOGRAM                     SYSTEM
 SET SUMMARY DISPLAY       BAR CHART                     OPTIONS
 TABULAR DISPLAY           CROSSTABS
                           STEP-WISE REGRESSION

                       YOU ARE NOW IN INVOKE MODE
                 YOU MAY SELECT ANY OF THE ABOVE OPTIONS

     COMMUNICATION WITH THE DATA ANALYSIS SYSTEM IS VIA THE
     DISPLAY POINTING DEVICE (LIGHT PEN OR STYLUS).  USE YOUR
     POINTING DEVICE TO TOUCH THE NAME (ON THE DISPLAY SUR-
     FACE) OF THE OPTION YOU WISH TO INVOKE.
     TO AID IN YOUR SELECTION NOTE THE FOLLOWING OPTIONS
     WHICH APPEAR WITH MOST DISPLAYS:
         HELP - PROVIDES A DESCRIPTION OF YOUR CURRENT STATUS
         TUTORIAL - PROVIDES A SUMMARY OF THE OPTION SELECTED
     TO OBTAIN ADDITIONAL INFORMATION BEFORE STARTING, TOUCH
     THE "OPERATIONS" & "SYSTEMS" OPTIONS IN THE TUTORIAL
     MENU ABOVE.
```

Figure 1



```
                         PLOT DISPLAY
        SET NAMES                  ATTRIBUTE NAMES
        SAWTELLE                   COPYOPTN
        COPIES                     SRVYOPTN
        FINNEGAN                   SORTOPTN
        NOTFINN                    CPUSCNDS
        NORMSORT                   ELAPTIME

        SAWTELLE                   ELAPTIME
 DISPLAY PLOT USING MAX & MIN ATTRIBUTE VALUES      X-AXIS   NUMBRIOS <--
                                                    Y-AXIS   ELAPTIME
 SELECT MAX & MIN ATTRIBUTE VALUES

         YOU MAY SELECT AN ATTRIBUTE FOR THE AXIS INDICATED
               YOU MAY SELECT ONE OF THE ABOVE OPTIONS
```

Figure 2

The desired attribute can then be selected for the capability about to be executed.

The actual disposition of the attribute once selected depends on the context in which the selection occurs. If it is in a preplot context, for example, the attribute name will flash along side the axis on which its data will be plotted. Figure 2 illustrates this selection process.

In Figure 2 the attribute NUMBRIOS of set SAWTELLE was selected for plotting along the X AXIS. Note that both the set and attribute names selected are displayed at the bottom of their respective menus. The small arrow at the sides of each menu when touched will cause a new set of names to be displayed.

Loading the data analysis data base

Data are entered into the DAS through a batch component called the Data Base Loader. The main function of this component is to transform the input data collection into an output data file organized to facilitate interactive data manipulation and display in the on-line mode. This latter data file is referred to as the Data Analysis Data Base (DADB).

In addition to forming the DADB, the loader also constructs a dictionary defining set and attribute names, identifies and transforms missing values to a unique form, provides a facility for selecting random subsamples of cases, and stores a description of the data collection for eventual retrieval and display in the interactive mode. This added information is provided to the Loader in the form of a data definition directory.

Creation and description of sets

The set formation facility in the DAS provides a mechanism for naming and preserving data relationships in the system. As the analysts use of the data becomes more and more qualified, he requires a mechanism to express these qualifications and name them. The set formation facility also serves the purpose of allowing the analyst to delete data items and cases from the body of data he is working with to provide a simpler analysis or to concentrate on specific variables. Set formation provides a method for recombining data sets and attributes to produce aggregate sets.

```
Command Expression   Set Specification Expression
 CREATE SET           A FROM S1, S2, S3,... SN

 WITH ATTRIBUTES:

 A1 = f (s1, s2, s3...sk)          ┐
 A2 = f (s1, s2, s3...sk)          │
 .                                 ├─ Attribute Specification Expressions
 .                                 │
 .                                 │
 Aj = f (s1, s2, s3,...sk)         ┘
 WHERE:
 s1 > s2 and s3 < s4               ]─ Membership Specification Expression
```

Figure 3—Set formation language

1. *Set Formation Process*

Set formation is achieved by the use of a language. This language, in turn, drives the data management system. The set formation command, although having a superficial appearance of a programming language, is a definitional command. It describes the characteristics of the resulting set.

In the present implementation of the Data Analysis System the user types into the terminal the command form to be executed. The system saves the command form for every set created and permits easy retrieval for editing and re-execution as well as set description. In addition, during the set creation process a set of basic summary statistics are computed for every attribute in the set created. These are also retrievable for review.

2. *Syntax and Semantics of Set Creation Language*

Every statement in the command language consists of four expressions:

—Command expression
—Set specification expression
—Attribute specification expression
—Membership specification expression

The manner in which these expressions fit together in a statement is represented in the paradigm in Figure 3.

Command Expressions

The command expression of the statement specifies that a set is to be created. This includes a variety of forms, i.e., FORM SET, FORM, CREATE, CREATE SET. The number of sets and their function in the statement is determined by the command scope.

Set Specification Expressions

Figure 3 is an illustration of the form of the set formation command. In the example, the "A" and "S_1, S_2, S_3, . . . S_n" are sets in the set specification expressions. "A" is a character string chosen by the user to be used as a name for the set to be created. This name will be added in the dictionary of set names. "S_1, S_2, S^3, . . . S_n" is a series of character strings representing names of sets already created and maintained by the system.

Attribute Specification Expressions

Attribute specification is accomplished by functional expressions involving attributes of the sets specified. The attribute specification expression of the command, that is, "$A_1 = f(S_1, S_2, S_3, \ldots S_n)$" in Figure 3 defines a character string "A_1" which is to be the name of an attribute of the set being created. It also specifies, from the source sets, which of their attributes are to be used in forming the object attribute.

The source set attributes to be used can be manipulated before being included in the new set. The mathematical functions allowed for this operation are the standard set available with FORTRAN compilers. In addition to these mathematical functions Boolean subexpressions are permitted to define alternative functional forms for an attribute. In this manner alternative functional forms can be used based upon the value set for specified attributes in each case.

Membership Specification Expressions

The membership specification in the command applies a restriction on membership. This specification limits membership by filing, in the object set, only those cases whose attribute values satisfy the Boolean expressions of membership.

Membership specification is accomplished by a Boolean expression of attribute values of the members of the sets specified. This expression follows a "WHERE" clause and results in the command being executed only on those cases of the specified sets where the conditions of the logical expression are found to be true.

3. *Formation of Subsets*

One of the uses of the set formation language is the creation of subsets. This is accomplished by the use of the membership specification expression. Cases for the subset are chosen on the basis of their attributes values satisfying the Boolean expression. Those cases whose values do not satisfy the expression are not included in the subset.

4. *Formation of Sets by Union and Intersection*

In addition to creating proper subsets the set formation language can be used to define sets that are formed from more than one parent set. This type of formation can be performed by either the union or intersection operation. When each case (or observation) of data is included in the master set (i.e., the DADB) at loading time a unique identifier is assigned. This identifier is perpetuated throughout all sets in which an observation may be defined as a member.

When the set specification expression contains more than one set name for the source set, a union or intersection specification is included. These processes are performed on the identifier attributes in the source sets and are used to reassemble data groupings in a building block fashion or to locate those observations contained in sets of data representing different characteristics.

5. *Set Summary Display*

As discussed earlier, when a set is created a group of summary statistics are computed for each attribute. The *set summary display* option permits the user to retrieve these statistics as shown in Figure 4.

By changing set names the user can selectively recall related attributes from other sets and easily compare the statistics. Figure 4 shows a case where the user has called up 2 attributes (CPUSCNDS, ELAPTIME) for 2 different sets (SAWTELLE, FINNEGAN). In this case FINNEGAN is a proper subset of SAWTELLE and the set summary facility is being used to compare the two sets.

Analysis of sets

The Data Analysis System allows the user flexibility in analyzing and viewing his data and its interrelationships by providing the following analytical packages:

- Two variable plots with curve fitting
- histograms
- barcharts
- cross tabulations
- stepwise regression

The following is a review of the above tools and their associated features.

Two Variable Plots

To obtain a two variable plot the user requests the *plot* option and chooses the two attributes for the respective axes of the graph (as shown in Figure 1, 2). The system automatically computes the grid using the minimum and maximum values of the chosen attributes. When the computed scattergram is displayed, the user receives, in addition to the graph, a set of statistics for each attribute. The statistics displayed include: *minimum data value, maximum data value, sample size, mean, standard deviation, variance.*

With the plot displayed, the user may rescale the grid to focus attention on different sets of ranges. This option is used, for example, when there are outliers or clustered data values.

Also, after the plot appears on the screen the user may choose to fit the data with a line. By choosing this option, the system computes the best fit of the data in the form:

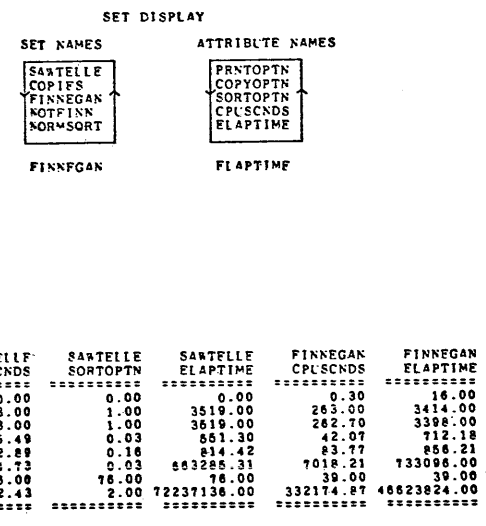
$$y = ax + b$$

The computed line is displayed along with values for a, b, and the statistics indicating "goodness of fit" (i.e., correlation coefficient, r^2).

Histogram

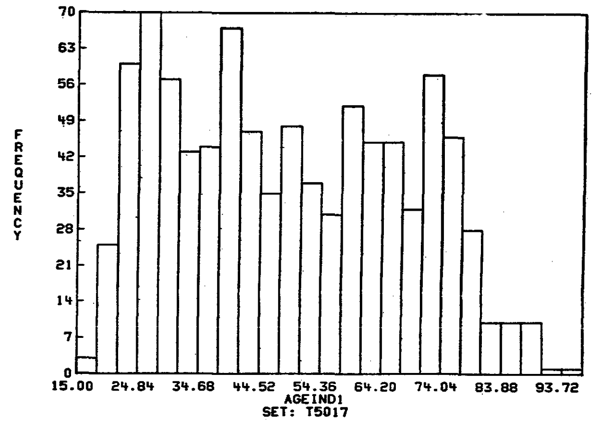
To compute and display the frequency distribution for an attribute having numeric values, the user indicates this to the system by choosing the *histogram* option and the attribute for which the histogram is to be drawn. The histogram is computed by using the maximum and minimum attribute values and the sample size. The computed histogram is displayed along with relevant statistics (see Figure 5).

When the frequency distribution has been displayed, the user has the option of rescaling the grid (if, for example, he detects an outlier) and redisplaying the histogram using these new values. As in the *plot* option, the user may recomplete and display the histogram as often as necessary to achieve desired results.



```
                SET DISPLAY

  SET NAMES              ATTRIBUTE NAMES

  SAWTELLE               PRNTOPTN
  COPIES                 COPYOPTN
  FINNEGAN               SORTOPTN
  NOTFINN                CPUSCNDS
  NORMSORT               ELAPTIME

  FINNEGAN               ELAPTIME

SETNAME      SAWTELLE    SAWTELLE     SAWTELLE    FINNEGAN    FINNEGAN
ATTNAME      CPUSCNDS    SORTOPTN     ELAPTIME    CPUSCNDS    ELAPTIME
=========> ==========  ==========  ==========  ==========  ==========
MINIMUM          0.00        0.00        0.00        0.30       16.00
MAXIMUM        283.00        1.00     3519.00      263.00     3414.00
RANGE          283.00        1.00     3519.00      262.70     3398.00
MEAN            25.49        0.03      661.30       42.07      712.18
STANDEV         62.89        0.16      814.42       83.77      856.21
VARIANC       3954.73        0.03   663285.31     7018.21   733096.00
SAMPSIZ         76.00       76.00       76.00       39.00       39.00
RMEANSQ     344682.43        2.00 72237136.00   332174.87 46623824.00
=========> ==========  ==========  ==========  ==========  ==========
```

Figure 4



```
*****************************************************************
MIN=            17.00      NO. OF INTERVALS=      25
MAX=            96.00      INTERVAL SIZE=        3.28
MEAN=           49.14      CHI-SQUARE=         229.63
VARIANCE=      358.52      DEG. OF FREE.=         22
STD. DEV.=      18.93      NO. OF POINTS=        905
*****************************************************************
```

Figure 5—Histogram display

Barchart

The *barchart* option allows the user to obtain a frequency distribution display for an attribute having discrete numeric or alphanumeric values. Along with the displayed barchart, the values for *relative* and *absolute* frequency are displayed similarly to the histogram. If the data is numeric, the following statistics are also displayed: *minimum value, maximum value, mean, number of observation, variance, standard deviation.*

Crosstabs

The *crosstab* option of the DAS gives the user the ability to display a 2-way crosstab. After indicating to the system that he wants the *crosstab* display, he chooses the two attributes which are to be cross tabulated. At this point the user has two options. He can let the system automatically generate the crosstab by computing the row and column values from the attribute data values or he can choose the values for the rows and columns explicitly.

Along with displaying the absolute frequency count for each cell, the user can indicate that he wishes the system to compute and display one of the following relative frequency percentages:

- percent of row total
- percent of column total
- percent of total sample size

The crosstab is displayed along with row and column totals, and relevant statistics for each attribute.

After the crosstab has been displayed, the user has the option of changing the row and/or column values, choosing a different relative frequency count to be displayed and redisplaying the crosstab with these new values. As in the other packages, new crosstabs may be computed and displayed as often as the user desires.

Stepwise Regression

The *stepwise regression* option is entered when the user indicates to the context manager to perform a regression analysis. Upon entering the package, he chooses a dependent variable and up to 29 independent variables to be included in the analysis. After the attributes have been chosen, the user has the option of setting his own tolerance limits for the regression (i.e., F-to-enter, F-to-remove, tolerance level & maximum number of steps), or of letting the system use its default values. The regression computation may proceed in two different ways: (1) The user can direct the system to perform the regression automatically letting the system choose the variables to be entered or removed at each step until the analysis is complete (i.e., one of the tolerance limits has been reached) or; (2) He can explicitly choose the variables to be entered or removed at each step, thus allowing him to have control over the variables to be entered or removed regardless of the tolerance levels.

This option is usually used when a definite model is being pursued.

At each step of the regression, or for the last step if the first mode is chosen, the following information is displayed:

- step number
- variable entered/removed
- standard error of the estimate
- multiple r
- analysis of variance table
- list of variables in the regression equation
- list of variables not in regression equation

When the regression is completed, a summary table is produced which summarizes the results of the regression at each step.

Also, at any point during the regression analysis, the user can view the following statistics are displayed for each variable in the regression:

mean, standard deviation, correlation with other variables, covariances with other variables

After the summary table has been produced, the user can display a tabular listing of the residuals and plot the residuals against any of the variables in the regression.

If the user discovers either at the end of the regression or during the regression that he has not chosen the correct variables, or that the tolerance limits should be changed, he can stop the regression, change the relevant information and restart the analysis. This can be done as often as is necessary to test the hypothesis.

IMPLEMENTATION

Reliance upon existing software

It was our intent, from the outset of the Data Analysis project, to capitalize upon as much existing software as possible. We felt that by doing this we could direct our efforts to those portions of the system which were unique and had not been addressed before.

The following are several highlights of this approach:

- *Implementation Language*

  The major part of the Data Analysis System was written in standard FORTRAN IV. The decision was made to use FORTRAN since most computer systems support it and thus provided options for portability.

- *DAS Language Processor*

  The command language developed for the Data Analysis System is translated using APAREL[6] (A Parse Request Language). APAREL is used as a series of commands in the standard PL/1 language. APAREL was chosen since it not only freed us from developing a language parser of our own, but also avoided the development of a unique translator for the DAS language.

- *Graphics Software*

  All of the graphics manipulation and display was accomplished using the Integrated Graphics System (IGS)[7] developed at The Rand Corporation. IGS is a series of graphic routines callable from languages with standard OS/360 linkage.

- *Analysis Packages*

  Whenever possible, the analysis packages provided by the system were taken from standardized, widely used packages and adapted to the interactive environment of the Data Analysis System. For example, this was done with the stepwise regression facility. For this we used the stepwise regression package of the BIOMED[8] Library (BMDO2R).

Hardware and systems

The prototype system has been implemented at The Rand Corporation on an IBM 360 model 65 computer under OS/MVT. The total amount of storage needed to run the system is 228K.

The hardware used for the graphical displays in the prototype system is the Rand Videographics System (VGS)[9] and a pointing device. The VGS consists of an interactive graphics console comprised of a cathode ray tube (CRT) and a keyboard. The pointing device can be either a data tablet or a light pen.

EXPERIENCE

Variety of users and responses

The Data Analysis system has been exposed to a variety of data bases, researchers, and research methodologies. These applications have included: the analysis of medical research data, studies in computer performance analysis, studies in the production of software and use in econometric and management sciences.

The response of these users to the system has been diverse which, in part, can be explained by the diversity of the data analysis process itself, i.e. there does not seem to be a single or predominant approach to analyzing a body of data. Significant variance in response seems also linked to both the analysts expectations from the data and the preconditioning of previous analysis experiences.

Two possible appeals of system

Some found the system most useful for data description and general "getting acquainted" with the data before applying in depth analysis techniques not supported by the system. For these users, the histogram, barcharting and plotting capabilities were used extensively to gather insights before applying more robust tests to the data.

Others found the system more useful for searching for unanticipated relationships in the data. These users found the system useful in forming hypotheses about the data and the phenomena under investigation. In some cases they felt that the data description capabilities of the system were better served by a batch processing system. These users said they generally began their analysis by performing data description on all variables routinely and were not disposed to performing this activity in an interactive mode. They felt this process was too routine for them to be interacting directly with the computer.

ACKNOWLEDGMENT

The authors would like to extend special thanks to Tom Wisniewski of RAND who was an integral member of the team that designed and implemented the Data Analysis System. Thanks are also due Carol Johnson of RAND who programmed the cross tabulation module of the system.

REFERENCES

1. Shure, G. H., *TRACE—Time Shared Routines for Analysis, Classification and Evaluation*, System Development Corporation, TM 2621/001/00, October 1966.
2. Miller, James R. III, *DATANAL: An Interpretive Language for On-Line Analysis of Empirical Data*, MITRE Corporation MTR-487, September 1967.
3. Hall, D. J., and G. H. Ball, "PROMENADE—An Interactive Graphics Pattern-Recognition System," In *Proceedings of IFPI Congress*, 1968, Edinburgh, Scotland.
4. Bowman, S., and R. A. Lickhalter, "Graphical Data Management in a Time-Shared Environment," *Proceedings SJCC*, 1968, Thompson Press.
5. Tukey, J. W., and M. D. Wilk, "Data Analysis and Statistics: An Expository Overview," *Proceedings, Fall Joint Computer Conference*, 1966, San Francisco, Spartan Books, Washington, D. C.
6. Balzer, R. M., and D. J. Farber, *APAREL—A Parse Request Language*, The Rand Corporation, RM-5611-1-ARPA, September, 1969.
7. Brown, G. D., and C. H. Bush, *The Integrated Graphics System for the IBM 2250*, The Rand Corporation, RM-5531-ARPA, October, 1968.
8. Dixon, W. J., Editor, *BMD—Biomedical Computer Programs*, University of California Press.
9. Armerding, G. W., and T. O. Ellis, *The Video Graphics Project*, The Rand Corporation, D-17788, September, 1968.

Quantification in a relational data system*

by NORTON R. GREENFELD

University of Southern California
Marina del Rey, California

The desire to express interrelationships between symbolic objects has been with us for some time, along with exploration of relationship systems which are operational in a computer. These systems coalesced under the term relational data systems (RDS), and a technology for dealing with this kind of data evolved. Relational systems have been through feasibility tests, experimental usage[1] and should become generally available to the computing community in the near future. The advantages which account for the expanding use of RDSs are a simple, formal definition which allows associative processing, extreme flexibility in both structure and use, an ability to be efficiently implemented, and a notation and conception which is not dependent upon any particular physical data representation.

Earlier stages saw the use of relational systems in computer graphics,[2] natural language systems,[3,4] general data management tasks,[5,6] and artificial intelligence research.[7,8] Experience with such systems has shown, however, that while the technology is adequate for small problems, for practical work it needs further development, especially in the area of quantification over relational forms.

By quantification here we mean substitutional quantification, as opposed to other kinds such as referential or objective quantification. These others involve general problems of deduction, a topic much too large for this paper. The limitation to quantification over describable entities leaves us none of the philosophical problems, but only ones of engineering. Substitutional quantification is still important: it is our only means of summarizing, and searching across large portions of a data base. Thus, "Is there some NCC paper longer than 20 pages?" or "Which NCC papers discuss data base problems?" are examples of simple queries with quantifiers. A more complex example might be: "How many NCC papers in this conference reference at least four NCC papers which reference at most three other NCC papers?" Methods for making queries like these efficient have been investigated as part of the REL project and are incorporated into the current REL English system.[9] This paper will discuss the nature of the problems and the types of solutions, along with some implications for relational systems design.

RELATIONAL DATA SYSTEMS

The following brief overview presents a vocabulary of relational systems. An RDS consists of a set of objects (often called entities, items, atoms, etc.) and a set of relations. Objects are primitive in the sense that they have no further structure, but can only be distinguished from one another and enter into interrelationships with each other. A relation is a set of ordered tuples (with the same number of elements in each). The degree of a relation R, written deg(R), is the number of elements in each of its tuples. A relationship is any single tuple in a relation, denoted by [R A1 . . . An], where R is the relation name and A1 through An form the tuple which is contained in R. Mathematically, given sets S1, . . . , Sn, an n-ary relation is a subset of $S1\times \ldots \times Sn$. Si is called the *i*th domain of the relation. Most RDSs single out the relations of degree 1 and call them sets or classes, and this convention will be followed here.

The primitive operations usually allowed in an RDS are:

1. creation and deletion of objects (or equivalently, the acquiring of the name of a heretofore unused object from a presumably infinite but fixed universe of discourse);
2. creation and deletion of relations;
3. the addition or deletion of a given relationship to a given relation;
4. a predicate which determines if a given relation contains a given relationship;
5. a retrieval function which, given a partially specified relationship, finds all relationships which match. A partially specified relationship means one in which some subset of the components of that relationship have been replaced by free variables.

Implementation questions in the past have dealt mainly with problems of efficient representation and access algorithms. The earliest representations were the LISP property list:[10] each atom had associated with it a list of the form (prop1 value1 prop2 value2 . . . propn valuen). Semantically, binary relations ⟨prop⟩ associate the atom with ⟨value⟩.

* This research was carried out as part of the REL Project, principal investigators Bozena H. Dostert and Frederick B. Thompson, at the California Institute of Technology. It is supported by Office of Naval Research Contract N00014-67-A-0094-0024, National Science Foundation Grant #GH-31573, Rome Air Development Center Contract #F30602-72-C-0249.

Operationally, given an atom and a "property," a linear search was conducted to find the value. Furthermore, only this particular access path is facilitated: to find the set of all pairs associated by a given property requires an inordinate amount of effort.

The need for symmetrical access in an efficient manner was first recognized by the LEAP originators.[8] This system basically used hash-coding and redundant data storage to achieve outstanding performance. The implementation saves binary relations (roughly) by hash-coding any two of the three elements involved (in a relationship) together to get a location to store the third element. Thus, the data is triply redundant and access by any two items is fast. Access by any single item was facilitated by further structure, essentially a threaded list through the hashed items. Other implementations of relational systems represent relations as vectors of n-tuples, or matrices.[4,12] This representation is symmetrical, relatively simple and slow (by comparison) for retrieval of single items, though fast for bulk retrieval or update.

The relational systems mentioned above have all implemented substitutional quantification by means of explicit generators. The meaning of the term is clear from the following algorithmic interpretation of the predicate calculus statement "For all x, P(x)":

1. generate first (next) object in the universe
2. if no more to generate, exit with value "true"
3. bind the variable x to the name of the generated object
4. evaluate P(x)
5. if value is "false," exit with value "false"
6. otherwise (value is "true"), continue at Step 1.

Thus the classic quantifiers, for-all and for-some, can be interpreted as rather simple algorithmic forms. Note that the interpretation of P in no way depends upon its being used inside the scope of a quantifier, an important system simplification. The diversity of desired quantifiers has required other forms, and to describe their interpretation as generators we will use LEAP as the prototypal example.

LEAP has items, sets, and triples. Items are atomic, and objects are either items or numbers. Sets are unary relations, distinguished in syntax and implementation. A triple is a notation whose first element is the name of a binary relation and whose second and third elements are the components of that relation.

LEAP is embedded in an ALGOL-like language and uses the following syntax for a triple: [A.O = V], signifying "Attribute of Object = Value." A retrieval can be requested by replacing any one or two elements of a triple by a variable name in an appropriate LEAP statement.

LEAP has one construction for a quantified expression, the ⟨loop statement⟩ whose syntax is

FOREACH ⟨associative context⟩ DO ⟨statement⟩

The ⟨associative context⟩ is a conjunction of Boolean expressions and retrieval triples. The operation of this construct can be described more easily in terms of a paraphrase:

FOREACH ⟨binding list⟩ SUCH THAT ⟨associative context⟩ DO ⟨statement⟩

where ⟨binding list⟩ contains those variables mentioned in ⟨associative context⟩ that are not already bound to some value. (This latter syntax is actually used in SAIL,[11] a descendant of LEAP.) The processing of this statement entails considering each of the conjuncts in the ⟨associative context⟩ in sequence, and using them both to filter values of variables already found and to retrieve possible values of other variables. The result of this process is a set of simultaneous values for all the variables in ⟨binding list⟩, each of which satisfy ⟨associative context⟩. The iterative execution of ⟨statement⟩ then takes place, with ⟨binding list⟩ variables being bound appropriately each time.

Note that in this LEAP operation, ⟨associative context⟩ is stated in terms of the primitive relational retrieval request of single relationships, and the system in fact implements the combined request by translating to that level. Both this imposed conceptual view and the implementation originate problems, which will be discussed in the next section.

PROBLEMS OF SIZE

The implementation of relational structures described above and the interpretation of quantifiers as generators have proved adequate in the past, but new applications with new requirements have revealed deficiencies. In most cases the problems are ones of efficiency, though there are also some conceptual implications.

The most immediate efficiency problem is one of size: today's data bases dwarf yesterday's. This large size means that a hierarchical memory environment is important, and dramatically influences the relevant operational characteristics of algorithms. Since the larger memory stores are slower and have more inertia than the smaller, primary stores, an algorithm's reference pattern to memory influences its elapsed time for execution. Particularly in the case of huge amounts of data, a slow execution time may be deemed equivalent to "impossible."

The LEAP hash-coded data structure was designed specifically with these problems in mind. For any single request to the data base, usually only one block of data need be brought from secondary to primary memory. That is, the algorithm references locations that are close to the desired data. None of the other implementations are as efficient for this purpose.

However, another important time for efficiency occurs during a quantified search, when a great deal of the data base must be checked. This is where the explicit generator method may be ineffective. To take the simplest case, for example, suppose we wish to find the image of a given class C, already known, under a given binary relation R. The LEAP statement would be

FOREACH x IN C AND [R.x = y] DO PUT y IN IMaGE.

Execution consists of generating the next member of the set C and binding that value to x, then doing an associative lookup on R and x, and for each value y found, adding it to the class IMAGE. In this simple case the best that LEAP can do is to reference some part of the relation R for each x value. If the data about R all fits into main memory simultaneously, then it will likely be brought in once and left there. However, when the relation R contains more data than will fit, a frequent occurrence, the reference pattern of this generator algorithm becomes unbearable.

To make this clear, suppose the routine has available K areas in main memory in which to place blocks of data, and suppose the data about relation R occupies M*K blocks. To find the value of R.x for a given, generated x, the system determines which block of R contains the information, and if that block is not already in main memory, brings it in. Since hash coding algorithms work best with uniformly distributed hashing functions, we can assume that any of the M*K blocks is equally likely to be required. Thus there is a probability of 1-1/M that a block must be brought in from the slow secondary memory for each value given x, that is, each member of the class.

The implications of this statement become more apparent when one considers that it is now easy to find data bases with classes containing from 1,000 to 100,000 members, and relations which are 10 to 100 times larger than available main memory. In these circumstances the hash technique will perform 1,000 to 100,000 input/output operations, while other methods, described below, require from 100 to 1,000.

The argument can be made, of course, that future generations of computers will have much larger primary memories and so this particular problem will disappear. What is not taken into account is that as our machine capabilities grow, the problems we wish to tackle will grow proportionally or even faster. In the case of memory size, larger capacity will make small problems extremely easy, but data bases are growing faster than our ability to manipulate them. The techniques mentioned here and elsewhere, and the foreseeable hardware developments, do not even allow us to deal effectively with the really immense data bases already available today.

SIZE PROBLEM SOLUTION

The solution to the memory reference problem has been rediscovered many times: group requests both spatially and temporally. For special purposes one can arrange the physical representation of the data and the accessing algorithms to maintain a locality of reference. In an RDS this means identification of those basic operations performed on relations by quantifiers and implementation with algorithms that have the appropriate characteristics. These operations become new "primitives" to the system and thus force new conceptualizations of the environment which have impact beyond the immediate reason for their introduction. Some of these implications will be mentioned in succeeding sections.

A complete list of primitive operations for a relational system falls beyond the scope of this paper, but some examples will convey the intent. Suppose R and S are n-ary and i-nary relations, respectively.

1. the permutation of R by k [k a permutation of the integers 1 through n] is that relation T such that if ⟨R1, . . . , Rn⟩ is in R then ⟨R(k1), . . . , R(kn)⟩ is in T.
2. the union of R and S (both n-ary) is that relation T which contains ⟨T1, . . . , Tn⟩ if that relationship is in either R or S.
3. the restriction of R by S is that relation T such that if ⟨R1, . . . , Rn⟩ is in R and ⟨R1, . . . , Ri⟩ is in S (and i< =n), then ⟨R1, . . . , Rn⟩ is in T.

These new operations subsume certain quantified statements, similar in effect to a Skolemization. The use of these operators allows system recognition of particular quantificational circumstances, and thus efficient handling. If, in fact, most use of quantifiers can be buried within such primitives, then extremely effective relational systems can be built and utilized over a variety of domains.

To consider the problem of efficient implementation, we use the restriction operator as a concrete example. In general, the algorithm must find all relationships in R whose initial components match some relationship in S. Because of the statement of the operation, an obvious implementation suggests itself. If a relation is stored as a vector of tuples, the classic technique of sort/merge works marvelously.

The analysis of input/output activity is enlightening. Suppose that R is stored on Rb blocks of secondary memory and contains Rr relationships. In most systems, the number of relationships per block ranges from 100 to 1,000. Similarly, S occupies Sb blocks and contains Sr members. Since we are considering the case in which R and S are very large compared with available main memory, the hash-code technique must input approximately Sr blocks of R (one for each element of S). The dual algorithm will require Rr blocks of S.

A sort of R needs about 2Rb*log(Rb) blocks to be input and output to a temporary file, and so the entire process of sorting R and S, and then merging the two will require

$$2Rb*\log(Rb)+2Sb*\log(Sb)+(Rb+Sb)$$

input/output operations. To compare these figures, assume for the moment that $Sb=Rb=n$ and $Sr=Rr=Kn$ (with K between 100 and 1,000). Then the sort/merge performs on the order of n*log(n) operations, while the hash-code requires Kn. Under these circumstances, assuming K at the minimum of 100 and only 10 frames of main memory available for the sort, the breakeven point between these two algorithms is $n=10\uparrow 24$. For any smaller relations, sort/merge is better, and for any larger ones the hash coding is again more efficient. Obviously even data bases of the near future will be much smaller.

There is another algorithm which beats both of the previously mentioned ones over a certain range of data base sizes. Called the SUBSET algorithm, it also requires the storage of a relation as a vector of tuples. Using the same notation as before, suppose the algorithm can ascertain that it has enough main memory available for K blocks of a relation. It can divide the relation R into subrelations, each of which occupies (K-2) blocks or less. The algorithm then iterates over these subrelations, and for each, brings it into main memory in its entirety. It uses one frame for input of S (one block at a time), and the second free frame for an output area. (There is a dual algorithm that subdivides S.) This algorithm obviously displays more knowledge and control of its environment, but in return for this complexity, the algorithm can be a factor of five or so better than the sort/merge algorithm. This algorithm requires Sb*ceil(Rb/K) (ceil means next-greatest-integer) input/output operations. This algorithm is of order n ↑ 2 in I/O operations (and CPU usage), and thus sort/merge is theoretically better in both. However, experimental results with the REL system prove that the SUBSET algorithm is the one to use for relations just a few times larger than available main memory. The REL data base manager actually computes the expected resource drain from each of the above algorithms and dynamically invokes the best one.

Notice that the physical representation of relations depends upon the algorithm used, and this in turn upon the predominant operations applied. As seems to be the case elsewhere, no general technique satisfies all requirements and circumstances. A well-designed interface to an RDS, however, with additional knowledge of the pecularities of the situation, may enable the RDS itself to select the appropriate representation.

OTHER KINDS OF GENERAL OPTIMIZATION

Once the door is opened on the processing of quantified relational expressions, several general kinds of possible optimizations emerge. These have little relationship to the problem of size discussed above.

The first, and somewhat obvious, optimization is the removal of constant expressions from quantified phrases. This "do-loop optimization" is well-known by compiler writers, and consists of moving expressions in a hierarchical iterative structure outward as far as possible. Thus the statement

```
FOREACH x IN set1 DO
    FOREACH y IN set2 DO
        If sister of x = mother of y THEN PRINT
          (x,y);
```

should be translated to:

```
FOREACH x IN set1 DO
    BEGIN temp ← sister of x;
        FOREACH y IN set2 DO
            IF temp=mother of y THEN PRINT
              (x,y);
    END;
```

In the particular case of relational systems, the primitives are small in number and have no side effects, and thus are ideal candidates for this kind of iterative optimization. There are basically two means for this. The first is a prepass over the quantificational form which identifies constant expressions and moves them, creating temporary variables as needed. The second method is akin to the operation of the Vienna Definition Language. Here each expression, when evaluated, replaces itself on the expression tree. Before every iteration, those expressions dependent upon the iteration variable get restored in the tree. Thus constants, relative to that particular iterative block, get evaluated only once, the first time through. This is the technique used by REL English language system, since the semantic operations are large and relatively few in number. With either method, though, there are large potential savings of execution effort.

The second general optimization area deals with ordering. This term covers several different problems and these especially pinpoint the need for considering relations as real entities with properties and primitive manipulatory operations. As an example of one type, consider the question of how to find the image of a class C under the composite relation RS. The system can either (1) find the image of C under S, then the image of the result under R, or (2) it can compute the composite relation RS and then directly find the image of C. This type of problem can be termed a "linear ordering" optimization. It is exactly here that the "explicit generator" view of quantification tends to hide the problem. The linear ordering optimizations deal with the associativity, commutivity, and distributivity of high order relational primitives. No research on linear ordering optimization has yet emerged in the literature.

Another type of ordering problem is one of simultaneous relational equations. In this case the system is asked to retrieve some objects that satisfy multiple relationship constraints. Thus, "find all x such that P(x)" and P(x) is a set of relationships, possibly including some internal quantification. These constraints, in fact, are used to guide the search. The problem concerns finding an optimum evaluation order, or, if the system allows simultaneous execution streams, finding the combination of parallelism and sequentiality that is optimum. The problem complications are that any particular atomic constraint can be used either to retrieve items or to filter already-found items, and there are differential cost functions of many parameters. A simple example, again from LEAP, is that of finding the sons of Bill:

```
FOREACH father.x=Bill AND sex.x=male DO
PUT x IN sons
```

The order in which the clauses are processed may have a marked effect on performance, since it is expected that there will be many more males than children of Bill. But the analysis is much more complex, depending in part upon the asymmetries of access. Simultaneity optimizations are particularly needed in the pattern matching programs contained in modern artificial intelligence languages[12] ("patterns" being partially specified relationships). Some research on the problem in the static, compile-time environment has been done,[13] and the Automatic Programming project at the University of Southern California's Information Sciences Institute[14] is currently investigating this area in the dynamic context.

CONCLUSION

This discussion of relational forms in quantificational situations has shown that efficient processing is possible, but at the cost of complexity. Each data representation and processing algorithm pair have a certain range of circumstances where that pair is appropriate. Relational Data Systems, then, can be built with one and only one such pair present, but this means a restricted domain of applicability. For a widely diverse domain, the system must be able to incorporate a variety of forms. To prevent difficulties in the use of such a collection, a stable, general interface is needed. This relational language should be independent of the particular representations chosen, and certainly should not hinder the use of some class of representations. It is the contention of this paper that the simple paradigm of quantifiers as explicit generators obstructs system recognition of some important situations and programmer recognition of the independent reality of relations.

The goal for future designers of Relational Data Systems consists of both a general descriptive language for relationship structures, and a system that implements those structures in appropriate ways, depending on the total environment. In the near future, of course, configuration decisions will be made by the programmer, but research into dynamic system decision-making may enable the programmer to ignore the problems of physical data representation, knowing that this will be done efficiently, and concentrate instead on the harder problems of logical structure.

REFERENCES

1. Final Report AUER-1776-TR-1, *Relational Data System Study*, Auerbach Corporation, Doc. #RADC-TR-70-180, July 1970.
2. Rovner, P. D., and J. A. Feldman, *An AMBIT/G programming language implementation*, MIT Lincoln Laboratory, Lexington, Mass., June 1968.
3. Thompson, F. B., P. C. Lockemann, B. H. Dosturt, and R. S. Deverill, "REL: A Rapidly Extensible Language System," *Proc. 24th National ACM Conference*, August 1966, pp. 399-417.
4. Kellogg, C. H., "A Natural Language Compiler for On-Line Data Management," *FJCC 1968*, pp. 473-492.
5. Levien, R. E., and M. E. Maron, "A Computer system for inference execution and data retrieval," *Comm. ACM* 10, 11, November 1967, pp. 715-721.
6. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Comm. of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387.
7. Green, C., "Application of Theorem-Proving to Problem Solving," Proc. *International Joint Conference on Artificial Intelligence*, D. E. Walker and L. M. Norton, Eds., Washington, D.C., 7-9 May 1969.
8. Feldman, J. A., and P. D. Rovner, "An Algol-based associative language," *Comm. ACM* 12, 8, August 1969, pp. 439-449.
9. Greenfeld, N. R., *Computer System Support for Data Analysis*, REL Project Report #4, California Institute of Technology, Pasadena, California, March 1972.
10. McCarthy, J., et al., *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, Mass., 1962.
11. VanLehn, K. A., *SAIL User Manual*, Stanford Artificial Intelligence Laboratory Memo AIM-204, Palo Alto, California, July 1973.
12. Bobrow, D. G. and B. Raphael, *New Programming Languages for AI Research*, Tutorial Lecture presented at the Third International Joint Conference on Artificial Intelligence, Stanford, California August 1973.
13. Hilbing, F. J., *The Analysis of Strategies for Paging a Large Associative Data Structure*, Ph.D. diss., Industrial Engineering, Stanford U., Stanford, Calif., March 1969.
14. Balzer, R. M., et al., *Domain-Independent Automatic Programming*, University of Southern California Information Sciences Institute Report ISI/RR-73-14, Marina del Rey, California, November 1973.

A public health data system

by JOHN C. PECK and FRANCIS M. CROWDER

Clemson University
Clemson, South Carolina

In mid-December, 1972 Clemson University became actively engaged in the technical development of a Public Health Data System for the Appalachian II District Health Department. Phase I of the implementation (ending June 30, 1973) has now been completed and has allowed both the Health Department and Clemson to more accurately measure development costs and benefits.

PUBLIC HEALTH PROBLEMS

Several factors which distinguish public health care from other clinic or hospital care are:

1. The size of the target population requires the necessity for extremely large volumes of data.
2. The widespread geographical distribution of the target population and public health service facilities makes record access extremely difficult.
3. The mobility of the target population is relatively high. A patient may require service in several facilities which have no records or knowledge of health care in other facilities.
4. Redundant data (name, address, sex, race, etc.) is captured not only in facilities at remote geographical locations but also within different programs within the same facility. In addition, many immunizations, lab tests and other services may be provided unnecessarily because of inadequate data access arrangements.
5. Standards for reporting of health care service are nonexistent; and because of the large number of health care providers who treat the patient, records are many times incomplete or ambiguous.
6. Manual systems typically present health care information in a source oriented sequence as opposed to a problem oriented sequence. Lab records are all stored together, immunizations together, prescribed medications together, etc. The problem oriented approach stores all information relating to each specific problem together. The results of a lab test, for example, must be interpreted in reference to the medications which are being taken. By storing information in the problem sequence, each patient complaint can be followed from beginning to end by any health care provider currently treating the patient.
7. Time series analysis of health related data becomes difficult with manual records. For example, a blood pressure of 140/80 may not be abnormal for some patients; however, if three months ago it was 120/60 then the physician may have ample reason for concern.
8. Investigation of trends in health care or disease becomes extremely laborious if not impossible with manual records.
9. Without a centralized data base drugs and/or physical conditions which conflict are difficult to discover. For example, many drugs should not be taken together or perhaps not be taken by pregnant women. If patients are being treated out of different facilities with different sets of records, conflicts of this nature may be difficult to determine.
10. Federal and state reporting requirements are becoming more time consuming and, therefore, provide less time for the health care professional to spend treating patients.

AUTOMATED HEALTH CARE

With an automated data file and remote access to the data, many of the problems inherent in manual records can be overcome. In addition, information processing and reporting procedures which were impossible to undertake with manual systems become simplified.

A brief description of the major points of implementation included in Phase I follows.

1. A master file consisting of patient related personal and socioeconomic data as well as pointers linking the patient to specific public health programs is created and maintained from cathode ray terminals (CRT's) located in each clinic. These terminals are connected to the Clemson University IBM 370/155 computer via telephone lines and can access and modify health related data under certain circumstances. As data is entered into this file it is edit checked for validity and flagged if incorrect or inconsistent so that the CRT operator may make corrections immediately. Records for approximately 100,000 persons in a two

```
PATIENT NUMBER 417-68-0390-                               DATE ENTERED 12/17/73
SOC SEC NUMBER 417-68-0390-
PATIENT NAME   SICK              BERTHA      I R          DATE BIRTH   06/16/50
MARITAL STATUS 1 MARRIED         SEX 1 MALE               RACE   5 OTHER WH.

ADDRESS STALL HEIGHTS                        CITY CLEMSON CITY
COUNTY  39        STATE 39   ZIPCODE 29631                CENSUS TRACT 112.00

TELEPHONE 803-656-4233       EMPLOYED 2 NO                EDUCATION 17

FAMILY INCOME AVAILABLE 00123                             NUMBER CHILD-HSHLD 01

NAME HEAD HSHLD NONE SINCE THIS  IS A TEST
LEGAL GUARDIAN  RECORD
ADDRS GUARDIAN  U R SICK

REFERRING SRCE  01 GREENVILLE COUNTY HEALTH DEPARTMENT
OTH AGY CONTACT 01 GVL CO HL DPT 02 SELF                    PROGRAMS ENROLLED
                04 AM  RED CROSS                          01 IMMUN   02 FAM PL
                                                          04 LEAD SC 05 CHILD D
REIMBURSEMENT   01 MEDICARE       02 MEDICAID             06 CHILD&Y 07 M&I
                04 AFDC-SCREEN    05 PLEA-SCREEN          08 HOME HS 09 VD
                                                          10 TUBERC  12 CRIP CH
```

Figure 1—Patient master

```
                                           ** ENTER B FOR EARLIER TESTS **
PAT. SICK                BERTHA     IR       **** LAB ****   PAT. NO 417-68-0390-
-DATE         06/20/73  04/30/73  12/01/72             06/20/73  04/30/73  12/01/72
WBC              43,1               10,3  URINE CAST    WAXY      HYAL      WAXY
HCT               42%       44%      24%   " CRYST     CA-OX     UACID     TRPOS
HGB                                 148%   " YEAST      MANY      RARE      MANY
RBC                     20 MIL    31 MIL   " BACT        FEW      RARE       FEW
MCV                                  001   " AMOR       RARE       FEW      RARE
MCH                                  003   " MUCOUS      FEW      MANY       FEW
MCHC%                               006%   " TRICH      MANY       FEW      MANY
BLOOD-FB                             064  EPITH      DRET&BLD    RENAL   VAGINAL
BLOOD GRP          O                      FECAL OCC      POS     TRACE       NEG
 RH           D+                          RPR          W REC    NONREC
~~RH TITER~~                              STOOL O&P     HOOK      WHIP
COOMBS             +                      G C CULT  ISOL CERV  NOT CERV
URINE SG      1.004                1.001  FTA            REC        BL
 " BLOOD       SPOS      MPOS          0  STS TITER  VD 0064   VD 0036
 " BILIR       MPOS     TRACE      TRACE  RUBELLA   1ST 0128
 " KET        TRACE      SPOS       MPOS  TOXOPLAM      0256
 " GLUC        MPOS     TRACE       SPOS  HB ELECT A1     10
 " PROT        SPOS      MPOS          0     "     A2     20
 " PH             8         7          7     "     F      30
 " WBC        21-49      6-10        1-5     "     S      40
 " RBC         6-10    50-100       6-
```

Figure 3—Laboratory display

county area are now accessible online. Expansion to a thirteen county area will be completed within the next eighteen months. (See Figure 1.)

2. A cross reference file, based upon the spelling of the last name, provides for quick and easy access to the master file data. When a patient master record is required the last name (and birthday, if known) is entered into the computer with the CRT. The computer then searches for all patient records whose last name "sounds" approximately the same as the name entered and displays them on the screen of the CRT. If the birthday was entered then only those patients with the sound-alike name and born on that day are displayed. (See Figure 2.)
3. Lab data related to a specific patient can be displayed in a time series format in which the last three lab reports are displayed side by side. Previous lab data can be displayed by backing up in the file using a "B" command on the CRT. (See Figure 3.)
4. Patient history data can also be displayed for any specific patient. This data provides a history not only for the patient but also for ancestors and siblings of the patient. (See Figure 4.)
5. Family planning regular checkup data is displayed in a time series format similar to lab data. A backup feature for checkups before the last three is also available. (See Figure 5.)
6. Immunization data can be displayed in a format which indicates the sequence number of each innoculation in a series along with the date of the innoculation. (See Figure 6.)

```
000203263      ELAINE       H SHOOK                 12/25/46
247226410      RUTH         F SHECK                 03/01/06
417680390      BERTHA     I R SICK                  06/16/50
800002208      JAMES        R SHOOK                 03/09/63
800002209      MELISSA        SHOOK                 11/03/67
800017561      SHANNEY      L SHOOK                 10/29/72
800044620      KEVIN          SCHAACK               02/14/55
800057027      RICHARD      R SHOOK                 06/09/66
800059297      LINDA        G SHOOK                 01/01/01
800066380      DAVID          SHOOK                 08/10/66
800066381      HEYWARD      A SHOOK                 10/03/68
800095001      SUSANA       C SHOOK                 07/18/68
```

Figure 2—Cross reference display

7. A "write" function is provided which allows the CRT operator to obtain a hard copy printout of any display on the CRT. In addition, special mark sense forms for the collection of additional immunization data can be prepared by the computer.
8. A "help" function is provided which lists all commands for the CRT operator in the event he cannot remember how to interact with the system. (See Figure 7.)
9. A "send" function is provided so that the CRT operator can communicate with the computer operator on the Clemson campus about any special procedures required on problems he might be experiencing.
10. Special data recovery programs have been written to guarantee integrity of data in the event of a device malfunction in which the data being collected online is destroyed.
11. Special security functions ensure that unauthorized personnel cannot access patient records. In addition, several levels of security clearance are provided for special data access functions. A predefined user identification and password must be supplied before the program can be initiated. Certain users can access all patient data while other users can access only a subset of the data. A clerk in the front office of the Health Department, for example, can enter and update patient records; however, he may not access medical records for the patient.

```
41/6[illegible]      [illegible]        BERTHA     I R   ADMITTED 06/30/73
IN EMERGENCY SICKIE         ALWAYS       Z  803-242-6160  REF 01
AGE ONSET 16 FREQ    DURATION 05 FLOW LIGHT    PAIN SLIGHT   INT BLD NONE
GRAV 12 PARA 02 AB 01 S 01 LIVING CHILD 07 IMM 06 SPON-A 0 IND-A 1
FETAL 1 NEONATAL 1 INFANT 1 PREM 1  LAST PREG 06/72
PREV CON FOAM        PRES CONT IUD          PT CHOICE PILL         WEL
PAST DISEASE HISTORY (1=POSITIVE, 2=SLIGHT, 3=MODERATE, 4=SEVERE)
1-ALLERGIES OR ASTH 1-CARDIOVASCULAR DI 1-RENAL DISEASE     1-VARICOSITIES
1-CONTRACEPTIVE COM 1-PHLEBITIS         1-CANCER OR TUMORS  1-DRUG SENSITIVITY
1-URINARY STRESS
SURGICAL & TRAUMA HISTORY
APPENDECTOMY        CERVICAL CONIZATION OVARIAN CYST        MASTECTOMY
ECTOPIC PREG.       GALLBLADDER SURG.   C-SECTION
FAMILY HISTORY (1=MOTHER, 2=FATHER, 3=PARENTS, 4=GRANDPARENTS, 5=SIBLINGS
           6=PARENTS&SIBLINGS, 7=OTHERS)
1-ALLERGY           1-ANEMIA            2-CANCER            3-DIABETES
1 HEART DISEASE     4-HYPERTENSION      3-KIDNEY TROUBLE    5-RHEUMATIC FEVER
1-DRUG SENSITIVE    6-OTHER - SEE MRS
SIGNIFICANT FINDINGS
EENT                TEETH               NODES               BREAST
CARDIAC             VARICOSITIES        SKELETAL            PELVIC
RECTAL
```

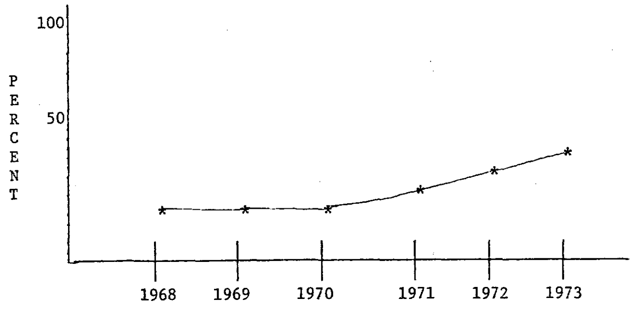
Figure 4—Family history display

```
PAT NAME BERTHA     I R SICK                PAT NO 417-68-0390-
DATE       08/01/73  06/20/73  04/30/73                08/01/73  06/20/73  04/30/73
BP S/D      174/110   178/111   180/110  LMP           03/28/73  03/28/73  03/28/73
HEIGHT         5-07      5-07      5-07    INTERVAL          30        30        30
WEIGHT          135       130       125    DURAT             05        05        05
BREAST TEND    BILI      BILI     RIGHT    AMT              MOD       MOD       MOD
  MASS        RIGHT     RIGHT     RIGHT  CRAMP                R         R         R
  DRAIN        LEFT     RIGHT     RIGHT    SEV              MOD       MOD       MOD
HEADACHE       MILD      MILD      MILD    FREQ           OCCAS     OCCAS     OCCAS
  FREQ         FREQ      FREQ     OCCAS  DISCH            HEAVY     HEAVY     HEAVY
  DURAT          06        10        07    FREQ            FREQ      FREQ      FREQ
DIZZINESS       MOD       MOD       MOD    COLOR          GREEN     GREEN     GREEN
  FREQ         FREQ      FREQ     OCCAS    SYMPTOM IR        IT  IR OD     IR OD IT
  DURAT          03        04        02  N.M.BLEED       SLIGHT    SLIGHT    SLIGHT
NAUS&VOM.       IRR       IRR        AM    TIME             M-C       M-C       M-C
  FREQ        DAILY      FREQ     OCCAS    FREQ              03        03        03
  DURAT          31        31        31    DURAT            2-3       2-3       2-3
EDEMA         HANDS     HANDS     HANDS  CONTRAC         C LOOP    C LOOP    C LOOP
  FREQ        OCCAS      FREQ     OCCAS    AMT
  MENS            R       UNR         R  RET/DATE      10/03/73  08/01/73  06/20/73
VARIC.EXT      BILI     RIGHT     RIGHT  CLINIC             010       010       010
  SEV          BOTH      PAIN      PAIN  PROVIDER           999       999       999
```

Figure 5—Family planning display

Phase II is currently in various phases of implementation and includes the following projects.

1. Scheduling subsystem to appoint patients to various clinics or health care providers subject to various constraints. Included in this subsystem is automatic notification of pending and missed appointments along with related reports for governmental and accounting requirements.
2. Data base management graphics display subsystem. A language allowing questions to be asked about statistics related to the data base will be developed. Answers to these questions will be answered in graphical form on a graphics display terminal. A typical question might be: "What percentage of the women between the ages of 15 and 35 in the family planning program over the last six years have also been enrolled in the venereal disease program?" The answer to the question might be a graph in the form:



3. Maternal Health subsystem.
4. Child Health subsystem.
5. Tuberculosis subsystem.
6. Veneral Disease subsystem.

```
                              IMMUNIZATION RECORD
PATIENT NAME BERTHA      I R SICK                    PATIENT NO. 417-68-0390
 POLIO       07/07/65 5  06/06/55 5  05/05/45 4  03/04/40 3  02/04/40 2
 DPT         07/07/65 4  06/06/55 4  05/05/45 4  03/04/40 3  02/04/40 2
 DT          08/09/73 4  08/01/71 4
 SMALL POX   05/05/60 0  01/04/40 0
 MEASLES     01/04/40 0
 MUMPS       03/03/42 0
 RUBELLA     02/02/41 0
 PENICILLIN  08/08/72 0
 INFLUENZA   07/07/70 0
```

Figure 6—Immunization display

```
COMMANDS:
    B     BACKUP IN FILE
    C     CHILD HEALTH DISPLAY
    CH    CHILD HEALTH HISTORY
    D     DELETE PREVIOUS MASTER RECORD DISPLAYED
    E     END SESSION
    F     FAMILY PLANNING DATA DISPLAY
    FH    FAMILY PLANNING HISTORY
    H     HELP REQUEST
    I     IMMUNIZATION HISTORY
    L     LAB DATA DISPLAY
    M     MORE RECORDS DISPLAY REQUEST
    P     PATIENT MASTER DISPLAY REQUEST
    S     SPECIAL IMMUNIZATION FORM
    SE    SEND MESSAGE TO OPERATOR
    U     UPDATE MASTER
    W     WRITE SCREEN INFORMATION ON PRINTER
```

Figure 7—Help display

APPROACHES TO IMPLEMENTATION

During the initial design phase of the Public Health Data System, several assumptions concerning the ultimate operating environment were made:

1. The program should be able to support an arbitrarily large number of terminals. Thus, modules must be designed to support re-entrant coding techniques.
2. Input/output logic should be confined to a central module so that commercially available data base management packages can be used if economically advantageous.
3. Program organization should lend itself well to conversion to a teleprocessing monitor system such as CICS (IBM), INTERCOM (GTE), or TASK MASTER (TURNKEY ASSOC.).

The development of the current operating package was carried out through several phases on Clemson University's IBM 370/155. The time sharing option (TSO) of IBM's operating system (OS/MVT) was used in the first phase of development for several reasons:

1. Program preparation and debugging was much easier in an interactive environment.
2. The main storage requirement for a non-time-shared implementation during program development would have been prohibitively expensive.
3. Only one data communications system was under development on the computer system so that costs for package systems could not be shared.

As developments of additional system features were begun and as the number of terminals serviced by the system increased, phase two of the technical development was initiated. The objective of this phase was to modify the system to run outside the time sharing region with a mixed environment message control program. Several factors prompted this change:

1. In the TSO environment each user had a separate copy of the program and was swapped between memory and disk storage as time was shared between users. With a large number of time sharing users the swap time became excessive and response time for transactions unacceptably large.
2. When an input or output operation is started by a time sharing program, no other time sharing program may be serviced until that I/O operation has been completed. Chaining through index structures and overflow areas using IBM's index sequential access method was locking out other time sharing users for the duration of the I/O.
3. Even with the carefully designed overlay structure implemented in the modular program, main memory allocated to the time sharing region became insufficient.

In order to move away from the time sharing environment, a driver program was written which provides the following functions:

1. Data blocks for each terminal required for re-entrant (multiple user) support must be maintained in memory or disk storage. A paging routine, using a least recently used algorithm, coordinates the swapping of data blocks between memory and disk storage.
2. Additional security provisions became available since the driver program controls and coordinates input and output to all terminals and users.
3. Special operation commands allow privileged users to assume the role of a master station and control the activity of other terminals and users.

The primary disadvantage of removing the system from the time sharing region was that a considerable amount of main storage must be dedicated for long periods of time throughout the day. Changes in the file organizations and access methods are now under way which will reduce the operating system data management storage requirements.

FUTURE DEVELOPMENT

Future developments depend upon many factors—the most important of which include continued acceptance by medical personnel and adequate funding. Eventual plans include the complete automation of the medical record to be stored in a centralized data base with computer terminals in all health care centers in a multiple county area. Records will be cross indexed by both problem and source so as to provide standardized and organized files which can be quickly accessed by health care providers.

Consultants in the medical profession will provide direction in the design of information displays and interpretation. This procedure will greatly help overcome many of the acceptance problems.

Additional computer hardware will eventually become a necessity if the system is to become completely operational. A central processor on the order of an IBM 370/145 will be dedicated to the application. A backup processor must be available to support the system in the event of a hardware failure on the main system. The Clemson University computer would serve as the backup computer without significantly degrading performance for campus users. Production will be restricted to the Health Department computer while development and systems support work will be restricted to the Clemson University computer.

During January 1974 Clemson University will acquire an IBM 370/158 and will run VS2. At that time all overlays and paging currently provided by the PHDS control program will be performed by the operating system. All ISAM files will be converted to VSAM which will provide faster access in the new environment.

A terminal monitor system capable of supporting multiple applications will be installed. IBM's time sharing option will still be used for program development but not for production support.

Terminal independent input/output subroutines, which make possible logical level programming for CRT's of arbitrary screen size and special features, will be written. Device dependent characteristics will be specified as constants and used with table driven logic keyed to terminal identification or line number.

The relationship now enjoyed by Clemson University and the Appalachian II Public Health Department will hopefully continue to benefit Clemson by providing areas for research in medical information systems and benefit the public health in South Carolina by providing the necessary technical expertise to successfully implement the automated systems.

Automated patient record summaries for health care auditing*

by ROBERT CHALICE, OLGA M. HARING, and RONALD HOCHSPRUNG

Northwestern University Medical School
Chicago, Illinois

Purpose and Objectives

At Northwestern University Medical Center, a computer printed summary report showing a patient's current condition appears to be especially useful in outpatient therapy. A printed summary report lends itself to the objectives of:

1. Displaying a patient's current condition in an *organized*, *up-to-date*, *accurate*, and *legible* form.
2. Auditing patient care in a uniform and consistent manner by processing the machine readable summary data.

With these objectives in mind, we are presently evaluating the summary report form shown in Appendix A, within a Cardiac-Pulmonary-Renal outpatient clinic at Northwestern University Medical Center. The care received by 240 patients who have printed summaries will be compared to the care received by a control group of 240 patients with nonsummarized charts.[1] Prior to each patient visit an up-to-date computer printed summary is inserted into the summarized chart. The summary report is then available as an aid to the physician who treats the patient. The assumption is that a higher quality of information will contribute to a higher quality of care. Preliminary studies by Janda[2] and Middlekauff[3] support this conclusion. Our current evaluation of the effects of the record summary on patient care began in October of 1972 and will extend over a three year period.

During the fall and winter quarters of 1969-70, the acceptability and utility of the Record Summary System to 50 fourth-year medical students and 20 physicians were evaluated.[2] The students were given timed tests in which they were required to retrieve specified information from both summarized and nonsummarized charts of comparable length. Students and physicians were given questionnaires on specific points and also were urged to criticize the whole program. The timed tests indicated that significant time saving and increased accuracy were achieved through the summarized record. Physicians stated that the summary was useful for patient care in its present form. They agreed without exception that the traditional record is frustrating and urgently needs reorganization.

In a pilot study carried out by Middlekauff in 1971, an attempt was made to determine not only the general feasibility and acceptability of the record summary but its effect on the quality of care given in the CPR clinic. The details of this study have been reported elsewhere.[3] Briefly, he selected 80 patients at random and summarized the charts of 40 of them. Each patient was assigned to one of ten attending physicians or one of ten fourth-year medical students, so that each of the 20 clinicians saw either four patients with a record summary or four patients without such a summary, for three months.

The quality of care was assessed in several ways. First, a set of standards based on the literature and agreement among Northwestern specialists was established for diagnostic and follow-up care. Second, each patient was asked to fill out a questionnaire containing eight questions after each of two visits to the clinic. Questions concerned symptoms and the patient's satisfaction with the care he received. Third, a record was made of the time the patient spent in the clinic, the number of suggestions made by the supervising physician and the number actually followed, and the changes in physical signs and results of laboratory tests between the first and the second visit. It was found that a larger proportion of standards were met for patients with summarized records than for those with nonsummarized records. Positive changes in physical signs and laboratory findings from the first to the second visit were more frequent and negative changes less frequent among patients with summarized records than among those without summaries. Questionnaires showed no difference in total satisfaction scores or in symptoms between patients with and without summarized records.

CONTENT OF THE PATIENT SUMMARY REPORT

Each summary report is computer printed and displays a patient's current condition in terms of:

IDENTIFICATION DATA
LAST HOSPITAL ADMISSION
PROBLEM LIST
VITAL SIGNS

* This project is supported by Grant Number HS00674-02 from the Department of Health, Education and Welfare.

CARDIAC-PULMONARY-RENAL DIAGNOSES
MEDICATIONS
DIETS AND OTHER THERAPY
PROCEDURES ORDERED
ROUTINE LAB TESTS
OTHER LAB TESTS
COMMENTS AND SUGGESTIONS

The Cardiac-Pulmonary-Renal diagnoses are individually listed, since these pertain directly to the specialty of the particular clinic under study. The nineteen routine laboratory tests have also been specially selected as those tests which should be regularly performed on a yearly basis for patients attending this clinic.

Problem oriented reporting is used whenever possible within the summary.[4] The summary, as shown in appendix A, contains a problem list, showing the status and disposition of all the patient's problems whether active, inactive, or resolved. Medications and diets are listed along with a reference to the problem for which they are prescribed. Procedures ordered are also tied to particular problems.

Computer Processing

The summary reports are currently maintained, updated, and printed by using a Control Data 6400 computer at Northwestern University. Utilizing a computer provides us with the following advantages:

1. Summary reports can be printed quickly on demand.
2. Frequent updates of summary data can be accomplished quickly and easily.
3. The data base is available in a machine readable form for high speed processing by other programs.

In addition to facilitating the maintenance of the data base and the display of information from it, computer programs can review the currency and consistency of patient data. A machine readable data base containing a particular patient's problem list, therapy, vital signs, medications and other relevant data makes it possible to methodically and meticulously check for anomalies in data, as well as responses to therapy, drug interactions, or other conditions that might warrant action. In addition, appropriate therapy can be suggested. Currently under consideration is a program to comment on the therapy given to hypertensive patients, according to accepted standards for treatment.

	Oct. 1972	Feb. 1973
Completely Abstract a New Summary	2 to 3 hr.	1 to 2 hr.
Keypunch and Print a New Summary	1½ hr.	1 hr.
TOTAL TIME FOR NEW SUMMARY	3.5 to 4.5 hr.	2 to 3 hr.
Update Last Printed Summary	20 min.	10 min.
Keypunch Revisions and Print New Summary	1 hr.	½ hr.
TOTAL TIME FOR UPDATED SUMMARY	1.33 hr.	.66 hr.

Figure 1—Times required to produce and update summaries

Today there is a demand by the government,[5] the medical profession, and also the public, that quality in medical care be assured. To do this it is essential that there be continual assessment of suitable methods for defining, evaluating and improving medical care.[6] Provision of a machine readable data base of patient summary information contributes to these goals of continual assessment, and the assurance of quality health care.

Initial Systems Design

Our desire from the start of this project has been to produce some objective statistical comparisons between the care received by summarized patients as opposed to the care received by nonsummarized patients. Because of this desire, we chose to quickly implement an initial system for maintaining and printing patient summaries in order to allow for an early comparison of care with the nonsummarized control group. In the meantime, a more sophisticated online system is under development, and will be discussed later. Because of its ease of implementation, a batch, card oriented system was chosen as the initial system for regular production of summaries. At the present time, we produce summaries for about 450 patients by using a batch oriented FORTRAN program. The summary data for a particular patient are presently maintained on an individual card deck for that patient. Updates of the summary data are made by changing the appropriate fields on the patient's data cards. Printed summary reports are produced by the FORTRAN program which reads and formats the summary data for output. The system is simple, and does satisfy the first objective of displaying up-to-date, well organized summary reports on patients.

Present Audit of Patient Care

The second objective of uniformly auditing patient care is accomplished by the batch oriented system in two ways. Prior to each patient visit the patient's last printed summary is updated by physicians and medical students who review the patient's chart. They may, at their option, provide suggestions to be included in the summary. These suggestions are then added to the last page of the summary for reference by the physician who treats the patient at his next visit. An audit of patient care by medically trained individuals is thus provided. Presently, the chart is the source of all summary data, although online collection of data in the clinic is being considered.

A second audit of care is also performed by the summary program as each patient's data is processed. Comments are automatically generated which call attention to important missing data, as well as important procedures which have not been performed at desired regular time intervals. Among the comments which can be generated are the following:

	Time Interval
1. PATIENT IS DUE FOR A COMPLETE YEARLY PHYSICAL EXAM. LAST PHYSICAL WAS ________.	1 yr.
2. PLEASE RECORD PATIENT'S BLOOD PRESSURE AT EACH VISIT.	
3. THE FOLLOWING PROBLEMS AND THEIR STATUS NEED REVIEWING:	½ yr.
4. PLEASE REVIEW CARDIAC-PULMONARY DIAGNOSES ON PAGE 4.	3 mos.
5. PLEASE ENTER CARDIAC-PULMONARY-RENAL DIAGNOSES ON PAGE 4.	
6. PLEASE COMPLETE CARDIAC-PULMONARY-RENAL DIAGNOSES ON PAGE 4.	
7. PLEASE FIND OUT IF THE PATIENT IS STILL TAKING MEDICATIONS LISTED ON PAGE 5.	6 mos.
8. PLEASE FIND OUT IF PATIENT IS STILL MAINTAINING DIETS AND OTHER THERAPY LISTED ON PAGE 5.	6 mos.
9. PLEASE ORDER: / routine test names /	1 yr.
10. PLEASE FIND OUT IF PATIENT HAS ANY DRUG ALLERGIES OR IDIOSYNCRASIES.	
11. PLEASE SPECIFY PROBLEMS FOR WHICH EACH OF THE FOLLOWING DRUGS IS PRESCRIBED.	

The time interval shown after some of the comments indicates that a comment should be generated when the date of a particular procedure or review lapses beyond the time indicated. Those of the above comments which are applicable to a particular patient are automatically printed on the last page of his summary, just after those comments that were manually provided by physicians and medical students.

The comment "PLEASE ORDER:" by which routine tests are suggested on a yearly basis deserves some special consideration since those routine tests which should be ordered must be listed by name. Furthermore, particular tests which need to be ordered must be grouped when appropriate and ordered as the multiple tests: SMA-6, SMA-14, and SMA-18. A comparison of costs is nedeed in order to decide when to order a particular multiple test instead of ordering a few components individually. The following rules are used by the computer program in suggesting that tests be ordered:

1. An SMA-6 includes electrolytes and is less expensive, so an SMA-6 is always ordered in preference to ordering electrolytes separately.
2. A full SMA-6 is always ordered in place of one or more components to be ordered individually because the cost of the SMA-6 is the same as the cost of ordering one component.
3. A full SMA-14 is ordered in place of three or more components to be ordered individually because the cost of the full SMA-14 is the same as the cost of ordering three components individually.
4. If ordering an SMA-14 is indicated by step 4. and any electrolytes are to be ordered, then an SMA-18 is ordered because it is less expensive than ordering an SMA-14 and electrolytes separately.

This kinds of cost consciousness is not generally attributed to physicians, so the summary program is designed to suggest that tests be ordered in a way that minimizes the cost to the patient. It should be mentioned, though, that the summary program does suggest that all nineteen routine tests be performed on a yearly basis, and therefore probably results in more laboratory tests being ordered for summarized patients.

The present card oriented system does satisfy the desired objectives and does illustrate the utility of a machine readable data base of patient summary information. The card oriented system is, however, unwieldy and is not very practical for large numbers of patients. Cards deteriorate; single fields on punched cards are difficult to update; and card processing is extremely slow when compared to using higher speed media such as magnetic tape, disk, or drum.

Future Online System

The advantages of maintaining a patient summary data base using online, random access storage, such as disk or drum, are numerous:

1. Data can be immediately updated or displayed online by using terminals.
2. A query language can be used to perform selective information retrieval.
3. Multiple programs with different purposes can make use of the data base simultaneously.
4. Higher speed processing is possible over larger volumes of data than could be processed by a card oriented system. In particular, rapid auditing of large numbers of patients is practical.
5. The data can be structured into tables, lists, chains, and groups which more accurately reflect the inherent relationships among data items.
6. Functions can be performed at a distance through the use of modems and data communications devices.

With these types of advantages in mind, we are currently developing an online data base management system referred to as DMS/11* using a Digital Equipment Corporation

*Presented at the Digital Equipment Corporation Users' Society Fall Symposium, Nov. 29, 1973, San Francisco.

PDP/11 minicomputer with the following configuration:

Central Processor
16K Bytes of core memory
3 Disk drives with 2.4 million characters/drive
1 High speed paper tape reader/punch
2 CRT display terminals
1 Operators' consol
1 Line printer at 100 lines/minute

The above equipment configuration is purposely limited in an attempt to illustrate that a minicomputer based, data management system is capable of the kind of data management usually reserved for large scale machines. We are interested in the minimal configuration that will provide an online patient summary system. However, the DMS/11 system under development is designed as a generalized data management system which can be applied to data bases other than the special purpose patient summary data base. The recommendations of the CODASYL data base task force are being followed to provide generalized file structures and generalized access techniques.[7]

PRESENT PERSONNEL AND OPERATING COSTS

Timings of manual procedures were taken in October of 1972, just after the project began, as well as a few months later in February of 1973. The personnel time involved in producing a new summary or in updating an old one is shown in Figure 1.

As can be seen, there was significant improvement between the two dates on which timings were surveyed. Under our present system the cost of personnel involved in producing a completely new summary is about $9.00, while the cost of producing an updated summary is about $3.00. The cost of computer time involved is about 20¢ per summary report printed, and is almost negligible in comparison to personnel costs.

Admittedly, the present system is a compromise design, quickly implemented to provide summary reports for evaluation purposes. Nonetheless, cost figures are provided to show the kind of costs that can be expected in using a batch oriented system to produce summary reports. We expect that by using an online system, we will be able to reduce these personnel costs by a factor of two. This estimate stems from the fact that the cost of data entry (i.e. keypunching) will be reduced by having the updater or abstracter enter data directly into a CRT display terminal. It is true that the online system requires a dedicated minicomputer, but it is our hope that adhering to a minimal computer configuration will keep the initial equipment costs within the financial reach of most medical institutions. As the system is developed, we hope that we will succeed in the objective of cost effectiveness, so that the benefits of the system can be realized practically.

REFERENCES

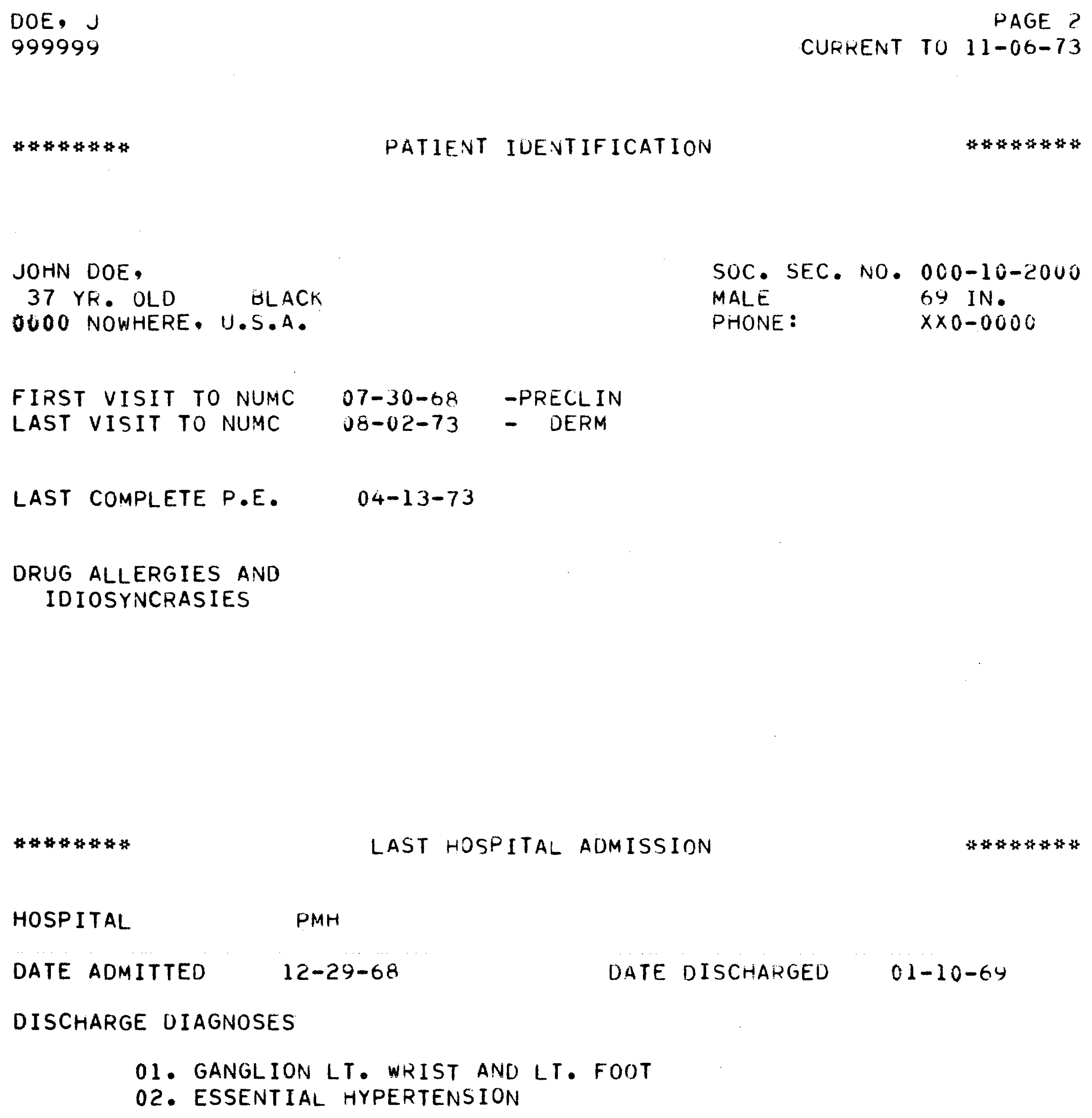
1. Haring, Olga M., "A Problem Oriented Record Summary for Use in the Clinic, *Chicago Medicine*, Vol. 76, No. 24, December 1, 1973, pp. 1003-1004.
2. Janda, Kathryn E., *A Method for Determining the Content and Form of a Physician Acceptable Medical Record*, M.S. thesis, Northwestern University, 1969.
3. Middlekauff, George W., *Evaluation of a Summarized Record System in an Out-patient Clinic*, Ph.D. dissertation, Northwestern University, 1972.
4. Weed, Lawrence L., *Medical Records, Medical Education and Patient Care*, Cleveland, The Press of Case Western Reserve University, 1969.
5. Sanazaro, Paul J., Richard L. Goldstein, James S. Roberts, David B. Maglott, and James W. McAllister, "Research and development in quality assurance: The experimental medical care review organization program," *New England Journal of Medicine*, Vol. 237, No. 22, November 30, 1972, pp. 1125-1131.
6. Payne, Beverly C., and Thomas F. Lyons, *Method of Evaluating and Improving Personal Medical Care Quality: Episode of Illness Study*, Ann Arbor, University of Michigan School of Medicine, February 1972.
7. CODASYL Data Base Task Group Report, Association for Computing Machinery, April 1971.

APPENDIX A

```
DOE, J                                                    PAGE 1
999999                                         CURRENT TO 11-06-73
NEXT VISIT TO       11-10-73                     PRINTED 02/27/74


                          NUCRSS-5

                   CLINICAL RECORD SUMMARY
                          NUMBER 5




   PLEASE LOOK AT THE LAST PAGE AND CONSIDER THE COMMENTS
   AND SUGGESTIONS.  FEEL FREE TO CORRECT OR COMMENT ON THIS SUMMARY.

                                THANK YOU,

                                DR. OLGA M. HARING
```

TABLE OF CONTENTS

```
UPDATED:
         -----
REVIEWED:
          -----
KEYPUNCHED:
            -----
```



```
DOE, J                                                              PAGE 2
999999                                                  CURRENT TO 11-06-73


********                  PATIENT IDENTIFICATION                  ********


JOHN DOE,                                     SOC. SEC. NO. 000-10-2000
 37 YR. OLD      BLACK                        MALE          69 IN.
0000 NOWHERE, U.S.A.                          PHONE:        XX0-0000


FIRST VISIT TO NUMC    07-30-68   -PRECLIN
LAST VISIT TO NUMC     08-02-73   -  DERM


LAST COMPLETE P.E.      04-13-73


DRUG ALLERGIES AND
  IDIOSYNCRASIES




********                 LAST HOSPITAL ADMISSION                  ********


HOSPITAL             PMH

DATE ADMITTED       12-29-68              DATE DISCHARGED     01-10-69

DISCHARGE DIAGNOSES

        01. GANGLION LT. WRIST AND LT. FOOT
        02. ESSENTIAL HYPERTENSION
```

```
DOE, J                                                          PAGE 3
999999                                            CURRENT TO 11-06-73


********          PROBLEMS IN ALL ORGAN SYSTEMS                ********
                               AND
                    VISITS TO MANAGING CLINICS


                       DATE     MANAGING   LAST
  PROBLEM              NOTED     CLINIC    NOTED     STATUS    DISPOSITION
  -------              -----    --------   -----     ------    -----------
01. HYPERTENSION      07-30-68   CPR      07-12-73   ACTIVE    RX ON P 5

02. ALCOHOLISM        05-07-69   CPR      05-21-71   ---       ---

03. LEUKODERMA        07-30-68   CPR      08-02-73   ACTIVE    RX ON P 5

04. SPRAIN,           08-30-71   ORTHO    ---        RESOLVED ---
     LT. ANKLE

05. DYSHIDROSIS,      08-13-68   DERM     08-13-68   ---       RX ON P 5
     LT. HAND

06. OBESITY, MILD     09-09-68   CPR      09-09-68   ACTIVE    ---

07. TINNITUS          10-13-72   CPR      10-13-72   ACTIVE    RX ON P 5

08. HEADACHES         09-10-71   NEURO    10-25-73   ---       RX ON P 5



TEMPORARY PROBLEMS


 A. INFLAMMATORY      04-19-73   DERM     08-02-73   ACTIVE    RX ON P 5
     SKIN CHANGES
```

```
DOE, J
999999                                                              PAGE 4
                                                          CURRENT TO 11-06-73


********                 VITAL SIGNS AT RECENT VISITS                 ********

  DATE     CLINIC    DOCTOR+STUDENT   WT    BP-SUPINE    PULSE-RAD   RESP   TEMP
  ----     ------    --------------   --    ---------    ---------   ----   ----

04-13-73    CPR      BROWN            174   130/92          ---       ---   98.4

01-19-73    CPR      LEVINE           176   140/110         ---       ---   98.8

10-13-72    CPR      WALKER           177   148/114 SIT     76        16    98.4

08-11-73    CPR      SMITH            175   140/118 SIT     ---       ---   98.6

07-14-72    CPR      BROWN            173   130/90          92        16    98.6

06-16-72    CPR      BROWN            175   150/104 SIT     68        18    98.0

06-09-72    CPR      BROWN            174   150/98          60        16    97.8

04-14-72    CPR      BROWN            179   160/120         72        16    98.0

01-21-72    CPR      HUNTER           175   150/108         72        16    98.6
********               CARDIAC-PULMONARY-RENAL CLINIC               ********

FIRST VISIT TO CPR      10-25-68
LAST VISIT TO CPR       04-13-73        CPR VISIT SCHEDULED FOR  11-10-73
  DOCTOR:                  BROWN
  STUDENT:                 ---

                                 DIAGNOSES
                                 ---------

HEART

   HEART               POTENTIAL HEART DISEASE

   ETIOLOGY            HYPERTENSION

   ANATOMY

   PHYSIOLOGY

   FUNCTIONAL CLASSIFICATION
   THERAPEUTIC CLASSIFICATION
   CLASSIFICATION REVIEWED

CIRCULATION            HYPERTENSION, ESSENTIAL

LUNGS                  NORMAL

KIDNEYS                NORMAL
```

```
DOE, J                                                      PAGE 5
999999                                              CURRENT TO 11-06-73

********                    MEDICATIONS                      ********

FOR                                    DOSE    RX                  RX LAST   RX
PROB DRUG AND SIZE                     SCHED   BEGUN    CLINIC     REVIEWED  TAKEN
---- -------------                     -----   -----    ------     --------  -----

1     HYDROCHLORO-       50 MG.        QD      06-72    CPR        08-10-73  ---
       THIAZIDE

1     RESERPINE          0.25 MG.      QD      04-72    CPR        08-10-73  ---

1     HYDRALAZINE        125 MG.       QD      10-72    CPR        08-10-73  ---

1     KCL LIQUID         15 CC.        BID     01-73    CPR        08-10-73  ---

5     VALISONE           0.1 PCT.      ---     08-68    DERM       ---       ---

7     MEPROBAMATE        400 MG.       QHS     10-72    CPR        ---       ---

8     FIORINAL           1-2           Q4-6HR  01-72    NEURO      10-25-73  ---

8     CAFERGOT                                          WESLEY     06-07-73

8     SAUSERT            2 MG          BID     06-28    NEURO      10-25-73

A     PSORALEN           10 MG         QD      08-73    DERM       10-25-73

A     KENALOG CREAM      .025          TID     05-73    DERM       10-25-73  ---

--- NO DIETS OR OTHER THERAPY ---


********                 PROCEDURES ORDERED                  ********

FOR                                                  DATE
PROB   PROCEDURE                          CLINIC    ORDERED     REPORTED(YES-NO)
----   ---------                          ------    -------     ----------------
8      BUN                                NEURO     07-26-73        NO
       ENT CONSULT                        NEURO     10-25-73        NO
```

```
DOE, J                                                   PAGE 6
999999                                       CURRENT TO 11-06-73

********                  ROUTINE TESTS                 ********

                  LATEST                   PREVIOUS
   TEST           DATE       RESULT        DATE       RESULT    CHANGES
   ----           ------     ------        --------   ------    -------

 1. CHEST X-RAY   04-06-73   NORMAL        07-12-71   NORMAL     NONE

 2. ECG           07-12-71   NORMAL        12-30-69   NORMAL     NONE

 3. URINE         06-28-73   NORMAL        07-12-71  ABNORMAL    BETTER


BLOOD

 4. RBC           07-23-73   14.8          06-28-73   4.76       NONE

 5. HGB           06-28-73   14.8          04-06-73   15.1       NONE

 6. CELL PACK     07-23-73   45            06-28-73   43         NONE

 7. WBC           06-28-73   4500          06-28-73   5100       NONE

 8. DIFFERENTIAL  06-28-73   NORMAL         ---        ---       ---


 9. VDRL          08-01-68   NONREACTIVE   ---         ---       ---

10. BUN           07-23-73   12            06-28-73   12         NONE

11. URIC ACID     04-06-73   7.3           08-12-71   7.5        NONE

12. CREATININE    07-23-73   1.5           04-06-73   1.20       NONE

13. FBS           06-28-73   101           04-18-69   100        NONE

14. 2 HR. PCS     10-25-68   52             ---        ---       ---

15. CHOLESTEROL   04-06-73   225            ---        ---       ---

16. SODIUM        06-28-73 * 137           04-06-73   134        WORSE

17. POTASSIUM     06-28-73   3.8           04-06-73   3.8        NONE

18. CHLORIDES     06-28-73   109           04-06-73   104        NONE

19. CO2           06-28-73   26.8          04-06-73   27.5       NONE
```

```
DOE, J                                                            PAGE 7
999999                                                CURRENT TO 11-06-73

********                        OTHER TESTS                      ********

       TEST            LATEST                    PREVIOUS
GROUP  NAME            DATE          RESULT      DATE        RESULT     CHANGES
-----  ----            ------        ------      --------    ------     -------

URINE/RENAL FUNCTION

       24 HR. URINE    11-12-68      2.1 MG.     ---         ---        ---
        VMA
       URINE           10-13-66      NORMAL      ---         ---        ---
        ELECTROLYTES

BLOOD CHEMISTRY ONE

       SGOT            04-06-73 *    69          08-12-71    28         WORSE
       SGPT            08-12-71      28          ---         ---        ---
       LDH             04-06-73      100         08-12-71    47         NONE
       ALK. PHOS.      04-06-73 *    95          03-09-71    9.4        WORSE
       GLUCOSE         04-06-73      95          ---         ---        ---
       SERUM LIPID     04-06-73      835         ---         ---        ---
       CPK             04-06-73      167         ---         ---        ---
       CALCIUM         04-06-73      9.8         ---         ---        ---

BLOOD CHEMISTRY TWO AND THREE

       BILIRUBIN,TOT   04-06-73      0.4         ---         ---        ---
       TOTAL PROTEIN   04-06-73      7.2         03-09-71    7.71       NONE
       ALBUMIN         04-06-73      4.3         03-09-71    5.02       NONE
       GLOBULIN        03-09-71      2.69        09-13-68    3.11       NONE
       GTT             10-25-68      NORMAL      ---         ---        ---
       BSP             10-13-66      NORMAL      ---         ---        ---

SEROLOGY/IMMUNOLOGY

       REITERS         08-01-68      NONREACTIVE ---         ---        ---
       LE PREP         01-15-69      NONREACTIVE 01-13-69    NONREACTIVENONE
       RIGHT FOREARM   08-30-73 *    ABNORMAL    ---         ---        ---
        BIOPSY

MICROBIOLOGY/CYTOLOGY

       URINE CULTURE   01-20-69      NEGATIVE    ---         ---        ---

NUCLEAR MEDICINE STUDIES

       RENOGRAM        10-13-66      NORMAL      ---         ---        ---
       BRAIN SCAN      06-21-73      NORMAL      ---         ---        ---
```

```
DOE, J
999999                                                      CONT≠D PAGE 7
                                                      CURRENT TO 11-06-73

********                     OTHER TESTS                         ********

       TEST              LATEST                        PREVIOUS
GROUP  NAME              DATE        RESULT            DATE        RESULT       CHANGES
-----  ----              ------      ------            --------    ------       -------

RADIOLOGICAL PROCEDURES

     LT. ANKLE           07-30-71    NORMAL            ---         ---          ---
     IVP                 10-13-66    NORMAL            ---         ---          ---
     CERVICAL SPINE      10-13-66    NORMAL            ---         ---          ---
     SKULL               10-13-66    NORMAL            ---         ---          ---
     SKULL               06-21-73    NORMAL            10-13-66     NORMAL      NONE
     ORBITS              06-21-73    NORMAL            ---         ---          ---

MISCELLANEOUS

     EEG                 11-29-71 *  ABNORMAL          ---         ---          ---


DOE, J
999999                                                         PAGE 8
                                                      CURRENT TO 11-06-73

********                COMMENTS AND SUGGESTIONS                 ********

  1. THE FOLLOWING PROBLEMS AND THEIR STATUS NEED REVIEWING:
             01. HYPERTENSION
             03. LEUKODERMA
             06. OBESITY, MILD
             07. TINNITUS
              A. INFLAMMATORY
                  SKIN CHANGES

  2. PLEASE REVIEW CARDIAC PULMONARY RENAL DIAGNOSES ON PAGE 4.

  3. PLEASE COMPLETE CARDIAC PULMONARY RENAL DIAGNOSES ON PAGE 4.

  4. PLEASE FIND OUT IF PATIENT IS STILL TAKING MEDICATIONS LISTED ON PAGE 5.

  5. PLEASE ORDER:
         ECG

  6. PLEASE FIND OUT IF PATIENT HAS ANY DRUG ALLERGIES OR IDIOSYNCRASIES.
```

An integrated health care information processing and retrieval system

by KEVIN C. O'KANE and RICHARD J. HILDEBRANDT

The Pennsylvania State University
University Park, Pennsylvania

INTRODUCTION

In this paper we present the design and some initial experiences with a computerized medical records system (called the CSAR System) currently in use in several departments of the Milton S. Hershey Medical Center of The Pennsylvania State University. The purpose of this work has been to develop a high-speed efficient information system for the storage, retrieval and dissemination of the total patient medical record.

Initially, we were concerned primarily with the development of a system for epidemiological research. That is, a system which could be used to isolate population cross-sections from extensive patient data bases at very high speeds. For example, we wanted a system which could, in a few minutes or less, identify from a population of a hundred thousand or more those patients whose records indicated fulfillment of criteria such as: "... between ages 25 and 30 with three or more pregnancies, type AB blood and a family history of cancer...."

Coupled with the need for a research system, we sought a design which could deliver an improved medical record into the day-to-day process of health care delivery. That is, a more legible, complete, accurate, accessible and standardized medical record at equal or lower overall cost. The selected system design would need to be capable of maintaining a patient's total medical record, not just a recent portion of it. The system would need to be able to flexibly access and portray on command all and only that information deemed relevant by the system's user. The design would need to be very efficient and economical so as to be able to store at reasonable cost historical data over a period of several years.

Furthermore, we wanted a system design which would incorporate both the above in a modular scheme so as to permit simultaneous but independent software development on many aspects. This implied a design consisting of two parts: First, a data base independent system nucleus which

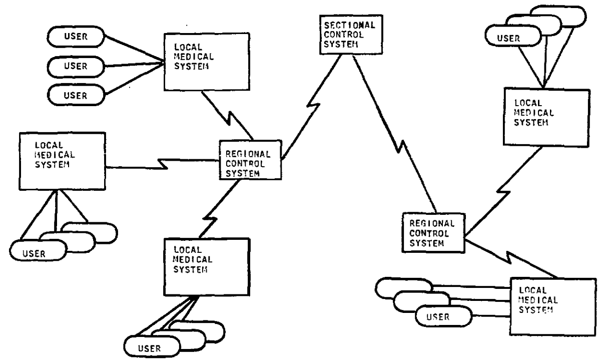


Figure 1—Distributed network of medical information systems

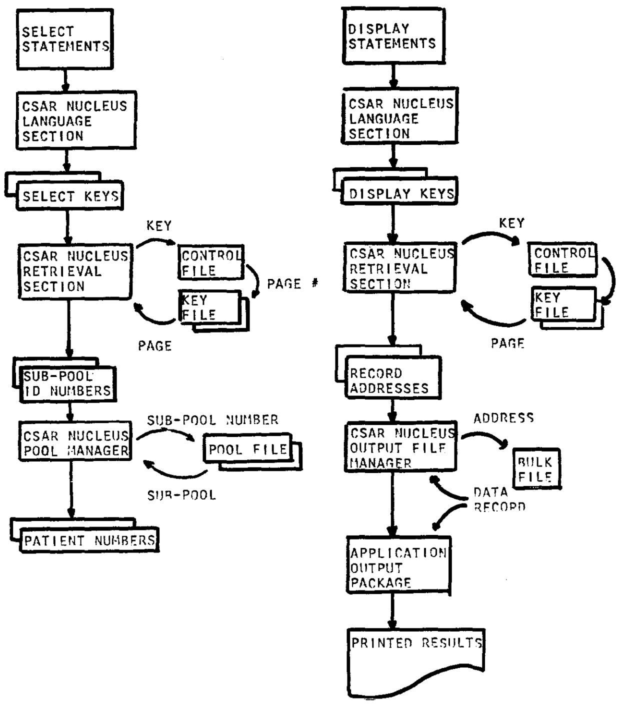


Figure 2—Basic system cycle

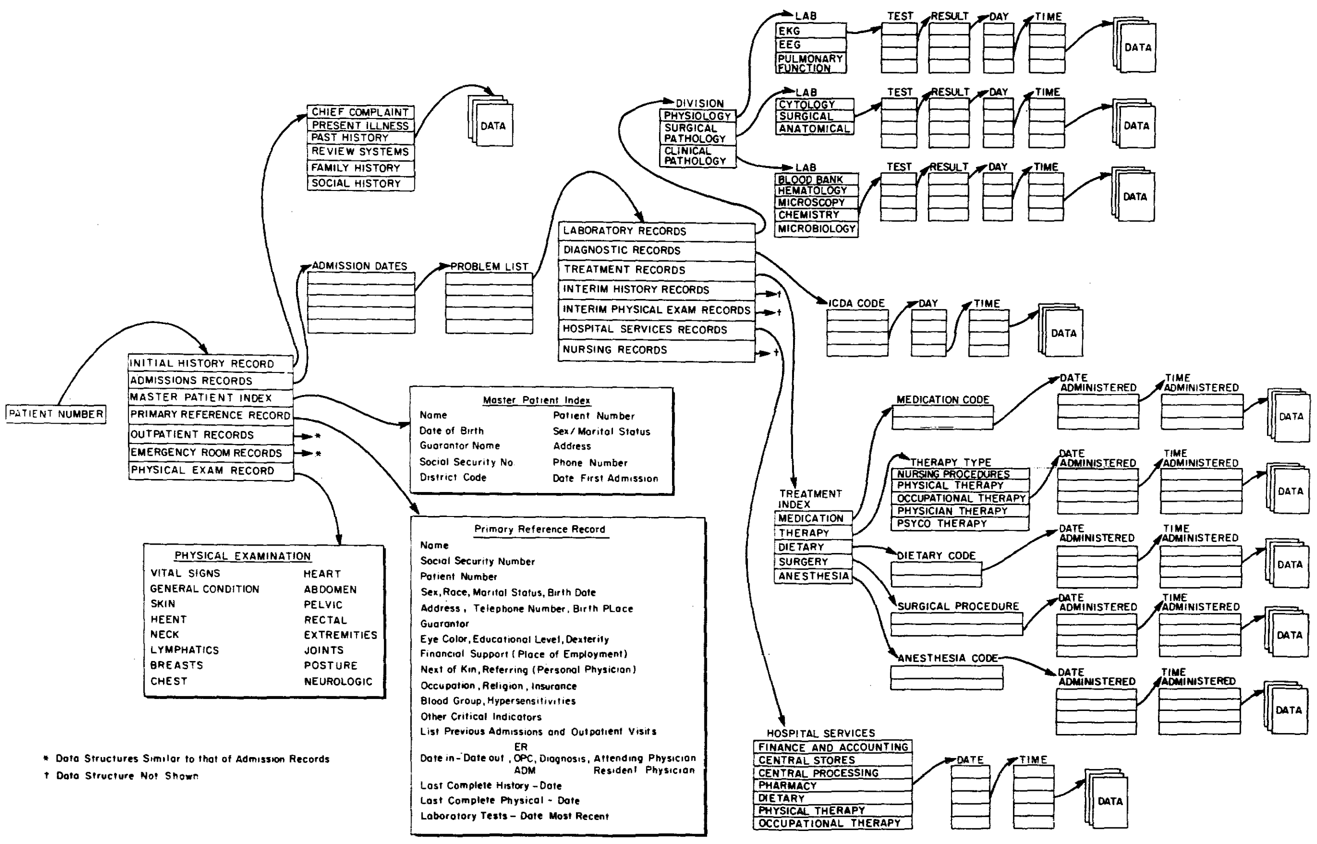


Figure 3—Structure of a medical record

would provide centralized input/output, file management, information processing and information retrieval services for other parts of the system. Second, applications interface packages which would deal with data base and user dependent aspects as printed output formats, update formats and so forth. In such a scheme, each data base dependent application could develop independently of the others within a common environment of conventions and structures.

TABLE I—List of Active Keys

Key-Word	Meaning
PROBLEM	ICDA DIAGNOSIS
PAP	PAP SMEAR RESULTS
USCR	URINE SCREEN RESULTS
ABSCR	ANTI-BODY SCREEN RESULTS
BLOOD TYPE	BLOOD TYPE
RPR	PROTEIN REACTION
RUBEL	RUBELLA TEST RESULTS
2HGL	2-HOUR GLUCOSE RESULTS
EDC	ESTIMATED DATE OF CONFINEMENT
HGB	HEMOGLOBIN TEST RESULTS
MEDICATION	DRUGS BEING ADMINISTERED
NAME	PATIENT NAME
RH	RH FACTOR
HCT	HEMATOCRIT TEST RESULTS
NEXT APPOINTMENT	APPOINTMENT LIST
ACTIVE PROBLEM	ACTIVE DIAGNOSIS
THERAPY	THERAPIES BEING ADMINISTERED

Another important design aspect is that of system expandability. That is, whether the system is capable of integrated growth without data base fragmentation. The design must assume at the outset that as the population serviced by the system grows, an upper limit to the capacity of any one computer implementation will be eventually achieved. At this point the population must be divided among more than one computer. As the number of such divisions grows, it must nonetheless remain possible for a user at any system to access data or perform searches at any other system or set of systems conveniently. That is, the user at an interactive terminal should not be aware of multiple machine data base segmentation.

DISTRIBUTED NETWORK OF MEDICAL INFORMATION SYSTEMS

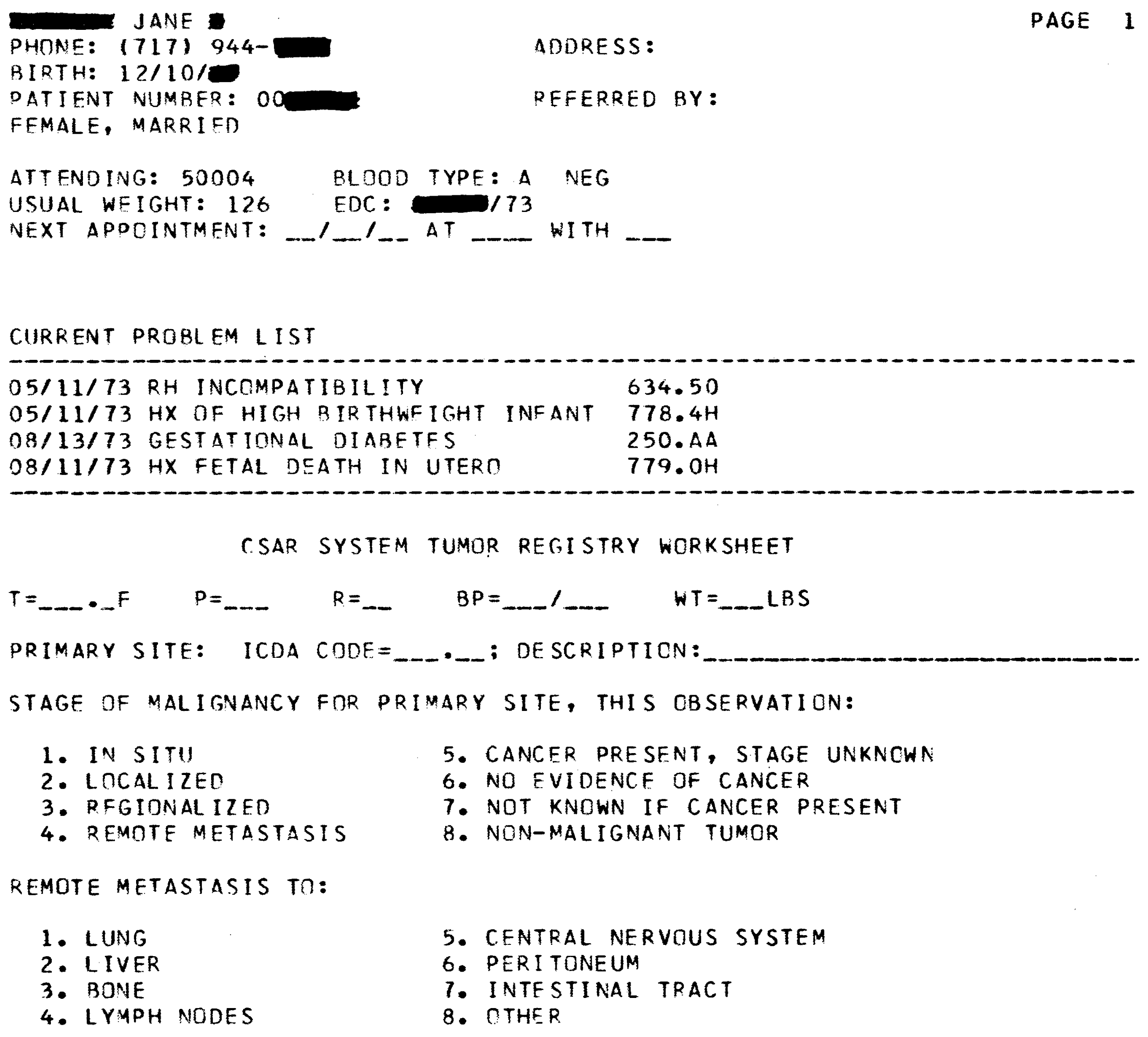
In order to fulfill the third specification mentioned above, that of expandability, a Distributed Network of Medical Information Systems is proposed. This is outlined in Figure 1. It will consist of many local medical information systems

```
$CSAR - SYSTEM FILES OPEN.
$CSAR - CSAR D.1 (OBSYS/TRSYS)
$CSAR - SUB-POOLS WILL BE ACCUMULATED.


 1 $NO LIST; /*DON'T LIST PATIENT NUMBERS AFTER SELECT STATEMENTS*/


 2 SELECT: IF: RH = NEG; /*NEGATIVE RH FACTOR*/
$CSAR - SUB-POOLS SELECTED:                                  1
$CSAR - PATIENT NUMBERS SELECTED:                                  59


 3 $AND;

$CSAR - SUB-POOLS WILL BE INTERSECTED.


 4 SELECT: IF: BLOOD TYPE = O;
$CSAR - SUB-POOLS SELECTED:                                  1
$CSAR - PATIENT NUMBERS SELECTED:                                  24


 5 $AND;

$CSAR - SUB-POOLS WILL BE INTERSECTED.


 6 SELECT: IF: PROBLEM = Y06.**; /*PREGNANCY ICDA CODE*/
$CSAR - SUB-POOLS SELECTED:                                 11
$CSAR - PATIENT NUMBERS SELECTED:                                  23


 7 $AND;

$CSAR - SUB-POOLS WILL BE INTERSECTED.


 8 SELECT: IF: EDC = 73/11/**; /*CONFINEMENT IN NOV 73, ANY DAY*/
$CSAR - SUB-POOLS SELECTED:                                 23
$CSAR - PATIENT NUMBERS SELECTED:                                   4


 9 $SORT: EDC; /*ESTIMATED DATE OF CONFINEMENT*/

$CSAR - PATIENT NUMBERS SORTED.
$CSAR - PATIENT NUMBERS SELECTED:                                   4


10 DISPLAY: CENSUS;

NAME                        NBR         SEX/MARITAL        PHONE              ATTND     BT/RH    WGT   EDC

[redacted] PATRICIA         00[redacted]  FEMALE, MARRIED    (   )    -         50[redacted]  O  NEG  134   11/[redacted]/73

[redacted] JOANNE E         00[redacted]  FEMALE, MARRIED    (717) 367-[redacted]  50[redacted]  O  NEG  105   11/[redacted]/73

[redacted] SUZANNE R        00[redacted]  FEMALE, MARRIED    (717) 533-[redacted]  50[redacted]  O  NEG  144   11/[redacted]/73

[redacted] CAROL            00[redacted]  FEMALE, MARRIED    (717) 273-[redacted]  50[redacted]  O  NEG  150   11/[redacted]/73
```

Figure 4—SELECT statement examples

```
 2 SELECT: IF: ATTENDING = 50[redacted]; /*ATTENDING PHYSICIAN CODE*/
$CSAR - SUB-POOLS SELECTED:                                   1
$CSAR - PATIENT NUMBERS SELECTED:                                 128


 3 $AND; /*"AND" NEXT PATIENT NUMBER LIST WITH CURRENT LIST*/

$CSAR - SUB-POOLS WILL BE INTERSECTED.


 4 SELECT: IF: PROBLEM = Y06.**; /*PREGNANCY ICDA CODE*/
$CSAR - SUB-POOLS SELECTED:                                  11
$CSAR - PATIENT NUMBERS SELECTED:                                 125


 5 $SET BASE; /*RETAIN THIS NUMBER*/
$CSAR - THE BASE IS:      1.250000E+02


 6 $AND; /*"AND" NEXT PATIENT NUMBER LIST WITH CURRENT LIST*/

$CSAR - SUB-POOLS WILL BE INTERSECTED.


 7 SELECT: IF: PAP = ****************; /*ACCEPT ANY VALUE*/
$CSAR - SUB-POOLS SELECTED:                                  25
$CSAR - PATIENT NUMBERS SELECTED:                                  83


 8 DISPLAY: PERCENTAGE; /*PERCENTAGE PREGNANT FOR 50[redacted] WITH PAP TESTS*/
$CSAR - CURRENT PATIENT NUMBER LIST IS 66.4% OF THE BASE.


 9 $XOR; /*"EXCLUSIVE OR" NEXT PATIENT NUMBER LIST WITH CURRENT LIST*/

$CSAR - SUB-POOLS WILL BE EXCLUSIVE OR'ED.


10 SELECT: IF: PROBLEM = Y06.**; /*PREGNANCY*/
$CSAR - SUB-POOLS SELECTED:                                  11
$CSAR - PATIENT NUMBERS SELECTED:                                 274


11 $AND; /*INTERSECT PATIENT NUMBER POOLS*/

$CSAR - SUB-POOLS WILL BE INTERSECTED.


12 SELECT: IF: ATTENDING = 50[redacted]; /*PHYSICIAN CODE*/
$CSAR - SUB-POOLS SELECTED:                                   1
$CSAR - PATIENT NUMBERS SELECTED:                                  42


13 DISPLAY: PERCENTAGE; /*PERCENTAGE PREGNANT FOR 50[redacted] WITHOUT PAP TESTS*/
$CSAR - CURRENT PATIENT NUMBER LIST IS 33.6% OF THE BASE.
```

Figure 5—SELECT statement examples



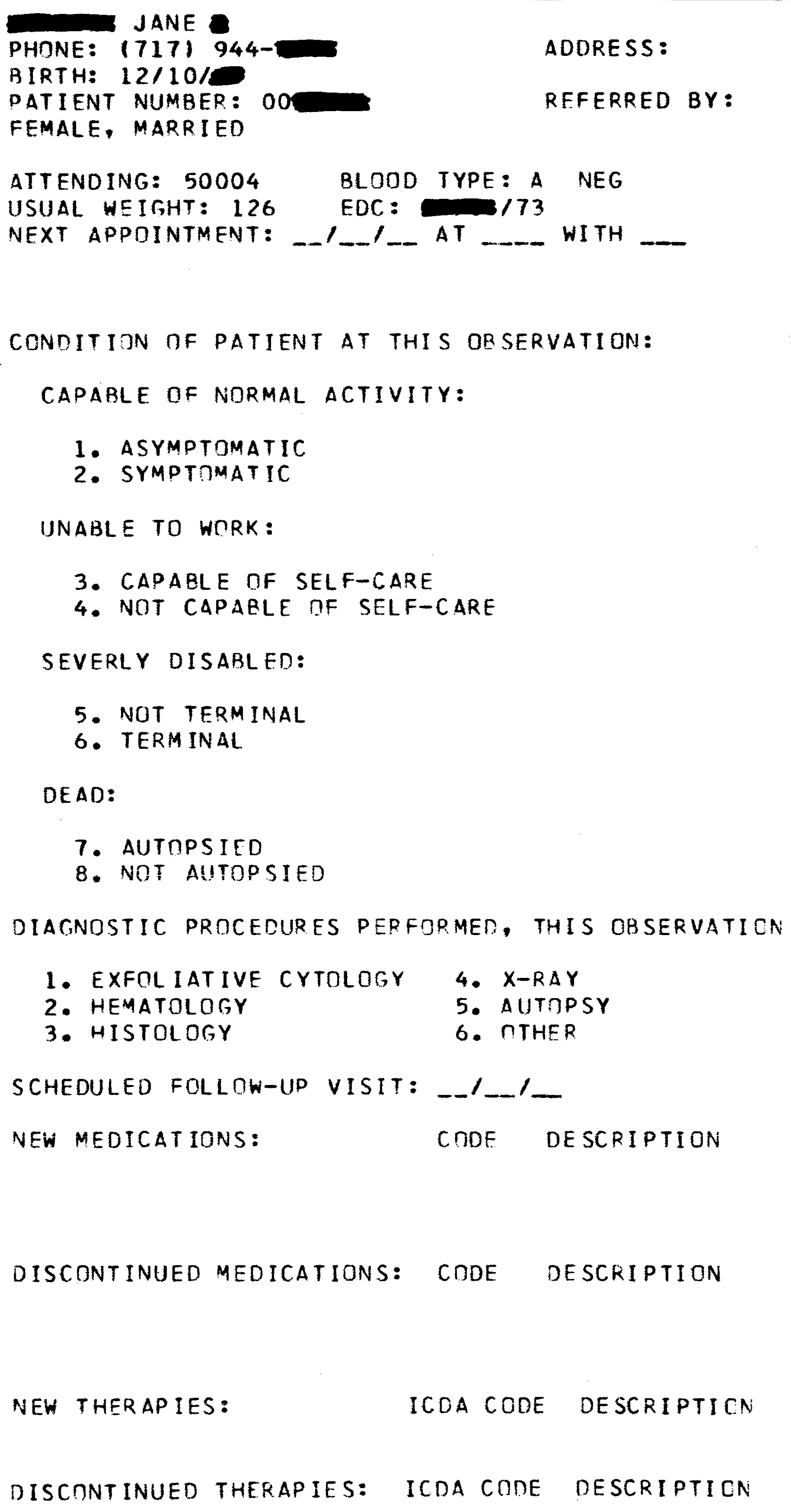
```
[redacted] JANE [redacted]                                          PAGE  1
PHONE: (717) 944-[redacted]               ADDRESS:
BIRTH: 12/10/[redacted]
PATIENT NUMBER: 0[redacted]               REFERRED BY:
FEMALE, MARRIED

ATTENDING: 50004     BLOOD TYPE: A  NEG
USUAL WEIGHT: 126    EDC: [redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


CURRENT PROBLEM LIST
-------------------------------------------------------------------------
05/11/73 RH INCOMPATIBILITY                 634.50
05/11/73 HX OF HIGH BIRTHWEIGHT INFANT      778.4H
08/13/73 GESTATIONAL DIABETES               250.AA
08/11/73 HX FETAL DEATH IN UTERO            779.0H
-------------------------------------------------------------------------

                CSAR SYSTEM TUMOR REGISTRY WORKSHEET

T=___._F     P=___     R=__     BP=___/___     WT=___LBS

PRIMARY SITE:  ICDA CODE=___.__; DESCRIPTION:___________________________

STAGE OF MALIGNANCY FOR PRIMARY SITE, THIS OBSERVATION:

  1. IN SITU                     5. CANCER PRESENT, STAGE UNKNOWN
  2. LOCALIZED                   6. NO EVIDENCE OF CANCER
  3. REGIONALIZED                7. NOT KNOWN IF CANCER PRESENT
  4. REMOTE METASTASIS           8. NON-MALIGNANT TUMOR

REMOTE METASTASIS TO:

  1. LUNG                        5. CENTRAL NERVOUS SYSTEM
  2. LIVER                       6. PERITONEUM
  3. BONE                        7. INTESTINAL TRACT
  4. LYMPH NODES                 8. OTHER
```

Figure 6—Tumor registry worksheet

connected centrally into regional control systems which could themselves, in turn, be connected into state-wide or larger networks. Each local facility will maintain its local data base which may be interrogated locally. In addition to this, however, each local facility can route requests to the central facility for access to information not stored locally. In turn, the central facility will poll, simultaneously, the proper set local facilities with the forwarded request. Replies will be collected and returned to the requesting station. The central control point need not be much more than a mini-computer with a suitable amount of telecommunications equipment and disk buffers for queueing of information. Alternatively, the central control point could be a major shared computer facility providing centralized facilities for many small medical centers.

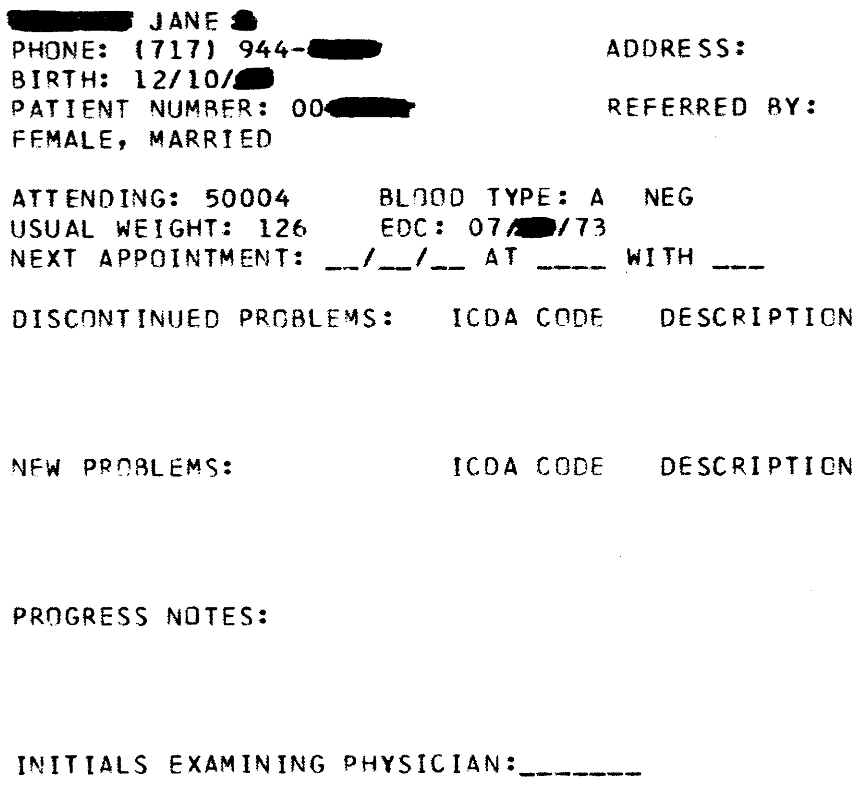
The great advantages of a design such as this are its simplicity flexibility and modularity. Control systems are easily designed. Flexibility and modularity is achieved across the network in that each local system can differ totally from each other system so long as it communicates its requests and replies according to a standard network protocol. This will permit a wide variety of local development to take place simultaneously.

Furthermore, if the central control points are regional shared resources, this design will permit smaller installations to access and benefit from computational and program library facilities which would otherwise be unavailable. The development of a distributed medical information network will enable individual institutions to specialize in designated areas of program development, library maintenance and so



```
[redacted] JANE [redacted]
PHONE: (717) 944-[redacted]                ADDRESS:
BIRTH: 12/10/[redacted]
PATIENT NUMBER: 00[redacted]               REFERRED BY:
FEMALE, MARRIED

ATTENDING: 50004     BLOOD TYPE: A  NEG
USUAL WEIGHT: 126    EDC: [redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


CONDITION OF PATIENT AT THIS OBSERVATION:

  CAPABLE OF NORMAL ACTIVITY:

    1. ASYMPTOMATIC
    2. SYMPTOMATIC

  UNABLE TO WORK:

    3. CAPABLE OF SELF-CARE
    4. NOT CAPABLE OF SELF-CARE

  SEVERLY DISABLED:

    5. NOT TERMINAL
    6. TERMINAL

  DEAD:

    7. AUTOPSIED
    8. NOT AUTOPSIED

DIAGNOSTIC PROCEDURES PERFORMED, THIS OBSERVATION

  1. EXFOLIATIVE CYTOLOGY    4. X-RAY
  2. HEMATOLOGY              5. AUTOPSY
  3. HISTOLOGY               6. OTHER

SCHEDULED FOLLOW-UP VISIT: __/__/__

NEW MEDICATIONS:             CODE   DESCRIPTION



DISCONTINUED MEDICATIONS:    CODE   DESCRIPTION



NEW THERAPIES:             ICDA CODE   DESCRIPTION


DISCONTINUED THERAPIES:    ICDA CODE   DESCRIPTION
```

Figure 6–2



```
[redacted] JANE [redacted]
PHONE: (717) 944-[redacted]                ADDRESS:
BIRTH: 12/10/[redacted]
PATIENT NUMBER: 00[redacted]               REFERRED BY:
FEMALE, MARRIED

ATTENDING: 50004      BLOOD TYPE: A   NEG
USUAL WEIGHT: 126     EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___

DISCONTINUED PROBLEMS:    ICDA CODE   DESCRIPTION



NEW PROBLEMS:             ICDA CODE   DESCRIPTION



PROGRESS NOTES:



INITIALS EXAMINING PHYSICIAN:_______
```

Figure 6–3

forth. The benefits of these efforts could be shared by all. Thus a medical facility in a small rural hospital could have access to the same tools at the same cost as a large metropolitan hospital. In general, this cost should be lower than it now is due to the elimination of duplication.

PROTOTYPE IMPLEMENTATION

At present, a prototype of a local medical information system is being tested. The prototype is written in PL/I and is being run on The Pennsylvania State University's IBM 370/165 research computation facility at University Park, Pennsylvania. The user population, at the Milton S. Hershey Medical Center in Hershey, Pennsylvania (103 miles distant), interact with the system via a Remote Job Entry system and a state-wide remote batch network. It is anticipated that work will commence shortly to transfer those portions of the system which are completed onto a small computer located in Hershey. Research and development, however, will still be carried out at University Park.

The basic system cycle is given in Figure 2. Figure 3 is an overview of the computer structure of the medical record. For purposes of explanation, user requests are divided into two classes: the "SELECT" statement and the "DISPLAY" statement. Others, however, are available but not considered here.

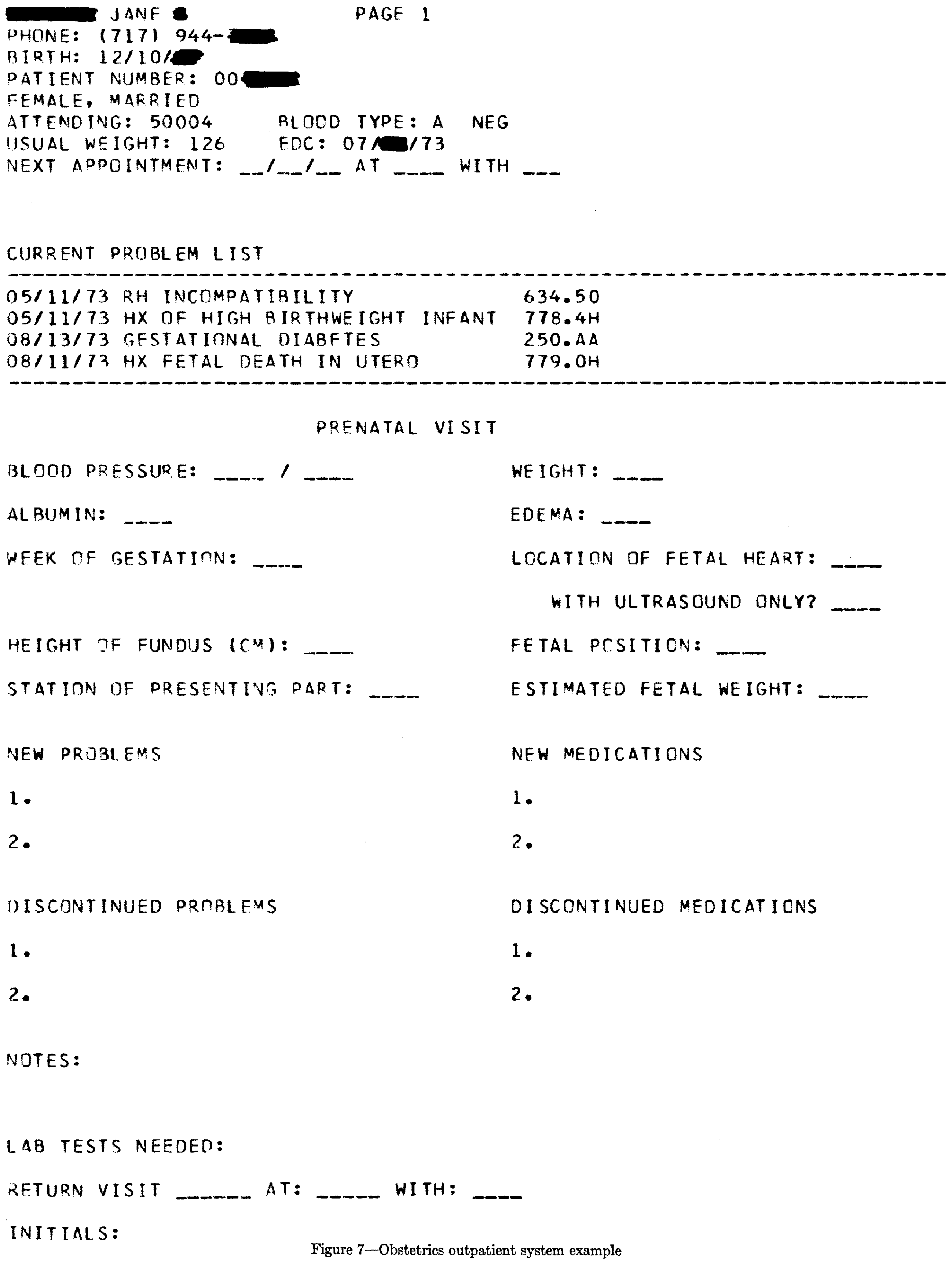
With the "SELECT" statement, the user specifies criteria for a population cross-section search. For example:

SELECT: IF: RH=NEG AND PROBLEM=Y06.0;

This request seeks the identity of all those patients with a negative RH factor who are pregnant (ICDA [1] Code). The result of this request is a list of patient identification numbers (hospital numbers) of those patients who fulfill the stated criteria. An example is given of such a request in Figure 4. The relational operators ">=", "<=" and "¬=" may also be used as can be seen in Figures 4 and 5. Asterisks are used to indicate that any recorded value for the selected field position will be acceptable. Table I gives a summary of those keys which are currently active for selection.

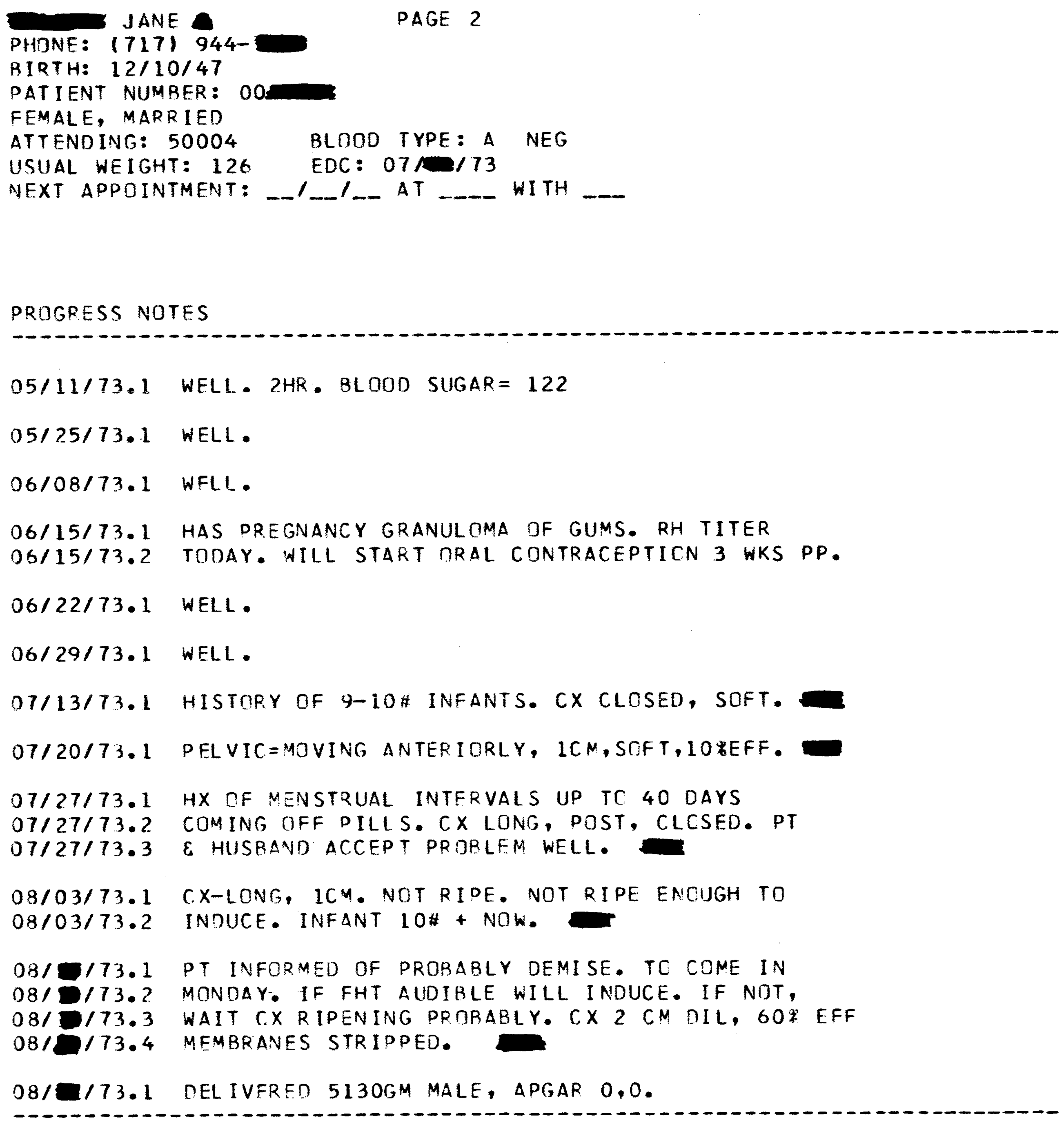
Pools of patient identification numbers are normally ordered by ascending value (default sort). This can be changed by the "SORT" statement as is shown in Figure 4. This is particularly useful when preparing patient charts for clinic visits. By selecting those patients who are scheduled for a given day, it is then possible to sort them by hour and by physician thereby greatly reducing clerical effort.

Having settled upon a list of patient identification numbers and having ordered them as desired, actual displays of data from patient records can be requested with the "DISPLAY" statement. Figures 6 and 7 give several examples of patient data displays. At present, there are three applications pack-



```
[redacted] JANE [redacted]             PAGE 1
PHONE: (717) 944-[redacted]
BIRTH: 12/10/[redacted]
PATIENT NUMBER: 00[redacted]
FEMALE, MARRIED
ATTENDING: 50004       BLOOD TYPE: A  NEG
USUAL WEIGHT: 126      EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


CURRENT PROBLEM LIST
------------------------------------------------------------------------
05/11/73 RH INCOMPATIBILITY                 634.50
05/11/73 HX OF HIGH BIRTHWEIGHT INFANT      778.4H
08/13/73 GESTATIONAL DIABETES               250.AA
08/11/73 HX FETAL DEATH IN UTERO            779.0H
------------------------------------------------------------------------

                          PRENATAL VISIT

BLOOD PRESSURE: ____ / ____            WEIGHT: ____

ALBUMIN: ____                          EDEMA: ____

WEEK OF GESTATION: ____                LOCATION OF FETAL HEART: ____

                                          WITH ULTRASOUND ONLY? ____

HEIGHT OF FUNDUS (CM): ____            FETAL POSITION: ____

STATION OF PRESENTING PART: ____       ESTIMATED FETAL WEIGHT: ____

NEW PROBLEMS                           NEW MEDICATIONS

1.                                     1.

2.                                     2.

DISCONTINUED PROBLEMS                  DISCONTINUED MEDICATIONS

1.                                     1.

2.                                     2.

NOTES:


LAB TESTS NEEDED:

RETURN VISIT ______ AT: _____ WITH: ____

INITIALS:
```

Figure 7—Obstetrics outpatient system example



```
[redacted] JANE [redacted]              PAGE 2
PHONE: (717) 944-[redacted]
BIRTH: 12/10/47
PATIENT NUMBER: 00[redacted]
FEMALE, MARRIED
ATTENDING: 50004        BLOOD TYPE: A  NEG
USUAL WEIGHT: 126       EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


PROGRESS NOTES
-------------------------------------------------------------------------

05/11/73.1  WELL. 2HR. BLOOD SUGAR= 122

05/25/73.1  WELL.

06/08/73.1  WELL.

06/15/73.1  HAS PREGNANCY GRANULOMA OF GUMS. RH TITER
06/15/73.2  TODAY. WILL START ORAL CONTRACEPTICN 3 WKS PP.

06/22/73.1  WELL.

06/29/73.1  WELL.

07/13/73.1  HISTORY OF 9-10# INFANTS. CX CLOSED, SOFT. [redacted]

07/20/73.1  PELVIC=MOVING ANTERIORLY, 1CM,SOFT,10%EFF. [redacted]

07/27/73.1  HX OF MENSTRUAL INTERVALS UP TC 40 DAYS
07/27/73.2  COMING OFF PILLS. CX LONG, POST, CLCSED. PT
07/27/73.3  & HUSBAND ACCEPT PROBLEM WELL. [redacted]

08/03/73.1  CX-LONG, 1CM. NOT RIPE. NOT RIPE ENOUGH TO
08/03/73.2  INDUCE. INFANT 10# + NOW. [redacted]

08/[redacted]/73.1  PT INFORMED OF PROBABLY DEMISE. TC COME IN
08/[redacted]/73.2  MONDAY. IF FHT AUDIBLE WILL INDUCE. IF NOT,
08/[redacted]/73.3  WAIT CX RIPENING PROBABLY. CX 2 CM DIL, 60% EFF
08/[redacted]/73.4  MEMBRANES STRIPPED. [redacted]

08/[redacted]/73.1  DELIVERED 5130GM MALE, APGAR 0,0.
-------------------------------------------------------------------------
```

Figure 7–2

ages in operation: (1) an Obstetrics Outpatient Clinic System; (2) an Obstetrics Labor and Delivery Information System and; (3) a Regional Tumor Registry System. Examples of pre-printed work-sheets and displays for these are given in Figures 6 and 7. Other systems, including a General Outpatient Clinic System which will be similar to the Obstetrics version, are in preparation.

SYSTEM DESIGN

Details of the system file structure and design are given in References 2 and 3. The overall design of the system consists of two parts. The first of these is a system nucleus which is largely data independent except for certain driver tables.

```
[redacted] JANE [redacted]              PAGE 3
PHONE: (717) 944-[redacted]
BIRTH: 12/10/47
PATIENT NUMBER: 00[redacted]
FEMALE, MARRIED
ATTENDING: 50004      BLOOD TYPE: A  NEG
USUAL WEIGHT: 126     EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


MEDICATIONS
-----------------------------------------------------------------------------
DATE      GENERIC NAME AND DOSAGE            CODE  DIRCTNS   DUR    RT DSCNTND

05/11/73 FERROUS SULFATE 300MG               26201 1TID             PO **/**/**
08/14/73 ORTHONOVUM 1/50                                            PO **/**/**
08/14/73 COLACE 100MG                        65206 1TID             PO **/**/**
08/11/73 RHOGAM                              82107                  IM **/**/**
-----------------------------------------------------------------------------

CURRENT PROBLEM LIST
-----------------------------------------------------------------------------
05/11/73 RH INCOMPATIBILITY                  634.50
05/11/73 HX OF HIGH BIRTHWEIGHT INFANT       778.4H
08/13/73 GESTATIONAL DIABETES                250.AA
08/11/73 HX FETAL DEATH IN UTERO             779.0H
-----------------------------------------------------------------------------

INACTIVE PROBLEM LIST
-----------------------------------------------------------------------------
05/11/73 PREGNANCY                           Y06.00                 08/11/73
07/13/73 LARGE BABY FOR DATES                777.41                 08/11/73
08/10/73 FETAL DEATH IN UTERO                779.9B                 08/11/73
08/11/73 SHOULDER DYSTOCIA                   656.8B                 08/11/73
08/11/73 4' PERINEAL LACERATION              658.3A                 08/11/73
-----------------------------------------------------------------------------


THERAPY LIST
-----------------------------------------------------------------------------
08/11/73  650.00  SPONTANEOUS DELIVERY
08/11/73  078.3C  REPAIR 4' LACERATION
-----------------------------------------------------------------------------
```

Figure 7-3

The second part consists of data dependent applications packages.

The system nucleus monitors and provides basic input/output services. It reads the system command language and, based upon entries in driver tables, constructs various population cross section keys ("SELECT" statement keys), patient data description keys ("DISPLAY" statement keys) and output report generation format tables. Nucleus routines retrieve patient numbers (for "SELECT" statement keys) and record addresses (for "DISPLAY" statement keys). The basic file structure is that of a blocked key file.[3]

The nucleus performs the logical operations (AND, OR, XOR) upon pools of retrieved patient numbers. In the case of record addresses (from "DISPLAY" statements), it retrieves the records and passes them to the proper applications package along with report format description tables.

During updates the nucleus monitors the optimal placement of patient records on bulk storage files. It automatically repositions patient records in order to maintain the complete record set for each patient on a single direct access disk cylinder thereby greatly reducing device arm movement. Other functions of the nucleus include sorting, timing and statistics collection. A built-in PL/I sub-set interpretive compiler permits dynamically entered user functions to interact with retrieved data.

Typically, applications interface packages are concerned with output print formats and updates. Routines vary in size and complexity depending upon the nature of the application. Data entered into a patient record as a result of any application package is available to any other package.

The system maintains four main files. These are: (1) a control file; (2) a key file consisting of pages or blocks of ordered keys; (3) a pool file containing sets (sub-pools) of patient numbers and (4) a bulk-file containing actual data records. Key-to-bulk file record address ("DISPLAY" re-requests) and key-to-patient number sub-pool ("SELECT" requests) are performed in the block structured key file. This works as follows:

At system initiation the control file is loaded. It lists the high and low key values for each block in the key file. Within each block keys are ordered from low to high. Associated with each key is a 32-bit number which is either: (1) a relative record address in the bulk-file or (2) the relative record address of a sub-pool of patient numbers in the pool file. A positive value indicates the first case and a negative value the second.

```
[redacted] JANE [redacted]                  PAGE 4
PHONE: (717) 944-[redacted]
BIRTH: 12/10/47
PATIENT NUMBER: 00[redacted]
FEMALE, MARRIED
ATTENDING: 50004          BLOOD TYPE: A     NEG
USUAL WEIGHT: 126         EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


                  DATE      TIME      STUDY     RESULT
                --------    ----      -----     ------
                01/12/73              RPR       NEG
                02/23/73              2HGL        122
                08/10/73              FIBR        793
                08/13/73    0700      GLUC        126
                08/13/73    0730      GLUC        176
                08/13/73    0800      GLUC        243
                08/13/73    0830      GLUC        274
                08/13/73    0900      GLUC        254
                08/13/73    1000      GLUC        163
                08/13/73    1200      GLUC         69
                08/13/73    0730      USG       NEG
                08/13/73    0800      USG       NEG
                08/13/73    0830      USG       NEG
                08/13/73    0900      USG       NEG
                08/13/73    1000      USG       NEG
                08/13/73    1200      USG       NEG
                -------------------------------------

                  DATE      WK   HCT    USCR   ABSCR
                --------    --   ----   ----   -----
                01/12/73    **   37.3   NEG    ****
                02/27/73    **   ****   ****   NEG
                06/15/73    **   ****   ****   NEG
                08/10/73    **   39.5   ****   ****
                08/13/73    **   40.1   ****   ****
                ----------------------------------
```

Figure 7–4

```
[redacted] JANE [redacted]              PAGE 5
PHONE: (717) 944-[redacted]
BIRTH: 12/10/47
PATIENT NUMBER: 00[redacted]
FEMALE, MARRIED
ATTENDING: 50004        BLOOD TYPE: A   NEG
USUAL WEIGHT: 126       EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___


             *** BLOOD PRESSURE BEYOND NORMAL LIMITS ***


                     BLOOD PRESSURE

          --------------------------------- 130
          .S                              .
          .   S   S                 S S   .
          .                               .
          . S   S     S S S S S S S     S .
          .         S                     .
          .                         D     .
          .                           D   .
          .D                              .
          .   D                           .
          . D   D D D D D   D D   D     D .
          .               D     D         .
          --------------------------------- 60


MAXIMUM SYSTOLIC: 130    MAXIMUM DIASTOLIC:  96
MINUMIM SYSTOLIC: 104    MINIMUM DIASTOLIC:  60
```

Figure 7–5

For retrieval, incoming keys are compared against the block high and low keys. If the incoming key lies outside the ranges of all, then nothing is retrieved. On the other hand, if the key lies within the range of some block, it is loaded (if not already resident). The block is searched in a binary manner. A list is developed consisting of the numbers (called registry numbers) associated with accepted keys. These are passed on to other parts of the nucleus.

The advantage of this type of retrieval technique is its speed and simplicity. Maximum time of search for an explicit key is, with a resident control file, independent of the size of the key file as bulk file. At worst case, the system must load a new page. Non-explicit searches vary in time depending upon the number of keys selected.

Experience indicates that, after an initial period of time, the rate at which new keys enter the system becomes linear and rises at a rate far less than corresponding increases in the size of the bulk file. This is also true of the pool file. The key file, at this writing, is 5 percent of the bulk file and pool file in size.

It is anticipated that the internal organization of blocks in key file will be restructured in the near future to that of m-way trees. This will reduce the amount of storage required. Further, it should decrease the number of page swaps.

PERFORMANCE

Presently, after 10 months of operation, there are about 500 patient records resident on the system. With the intro-

```
[redacted] JANE [redacted]            PAGE 6
PHONE: (717) 944-[redacted]
BIRTH: 12/10/47
PATIENT NUMBER: 00[redacted]
FEMALE, MARRIED
ATTENDING: 50004     BLOOD TYPE: A  NEG
USUAL WEIGHT: 126    EDC: 07/[redacted]/73
NEXT APPOINTMENT: __/__/__ AT ____ WITH ___



  DATE     TIME    WK    BP       ED   WT   AL   FH   FD   PF   ST   SZ   EXM
--------  -----    --  -------    --   ---  --   ---  --   --   --   --   ---
01/12/73  **:**    14  130/ 80    00   127  **   LMS  **   **   **   **   VGS
02/09/73  **:**    18  110/ 70    00   131  00   LMS  17   **   **   **   HCG
03/09/73  **:**    22  120/ 75    **   133  00   RLQ  18   **   **   **   DRH
04/06/73  **:**    26  110/ 70    00   141  TR   RLQ  18   **   **   **   DRH
05/11/73  **:**    31  120/ 68    00   148  00   RLQ  26   VT   FL   **   DRH
05/25/73  **:**    34  104/ 68    00   150  TR   LLQ  27   VT   FL   **   DRH
06/08/73  **:**    36  110/ 70    00   152  00   LLQ  31   VT   FL   **   DRH
06/15/73  **:**    37  110/ 64    00   152  00   RLQ  31   VT   FL   **   DRH
06/22/73  **:**    38  110/ 60    00   153  TR   LLQ  30   VT   EN   **   DRH
06/29/73  **:**    38  112/ 70    00   154  00   LLQ  32   VT   FL   **   DRH
07/06/73  **:**    40  112/ 70    00   154  00   RLQ  37   VT   DP   **   DRH
07/13/73  **:**    41  108/ 60    00   156  00   RLQ  35   VT   DP   **   DRH
07/20/73  **:**    42  108/ 70    00   156  00   LLQ  42   VT   FL   44   WJM
07/27/73  **:**    43  126/ 96    00   158  00   LLQ  34   VT   FL   **   DRH
08/03/73  **:**    44  120/ 90    TR   160  00   RLQ  38   VT   FL   **   DRH
08/10/73  **:**    45  110/ 70    00   158  00   NHS  41   VT   FL   45   DRH
------------------------------------------------------------------------------
WK = WEEK OF GESTATION;                      FD = HEIGHT OF FUNDUS(CM);
ED = EDEMA;                                  PF = PRESENTATION OF FETUS;
WT = MATERNAL WEIGHT;                        ST = STATION;
AL = ALBUMIN;                                SZ = FETAL SIZE(X100 GMS);
FH = POSITION OF FETAL HEART;                EXM = EXAMINER;


$CSAR - END OF SUMMARY FOR:                             [redacted] JANE [redacted]
```

Figure 7-6

duction of the recently completed Regional Tumor Registry Information System package, this is expected to reach 5,000 in the next 12 months. At present, the cost per patient per month is about $0.12 using an IBM 3330 Disk Storage device (this includes cost of the pack and drive plus shared costs of the controller and channel). The cost per patient record printout varies with respect to the size of the record, but is generally on the order of $0.05 to $0.15 for a moderately large record. Retrieval ("SELECT" statements) times vary upon the amount of patient numbers retrieved and are not significantly influenced by file size (this is characteristic of block structured techniques). In general, between 500 and 750 patient numbers per second can be retrieved.

CONCLUSION

We believe that the present system design can be exploited to handle the entire patient medical record economically. Having completed work on the basic system nucleus, several applications packages are under development. One such project is in the area of billing and finance. By including an additional flag in the field for each chargeable item or service, the system can be used to produce an integrated patient account statement. Another project is in the area of developing an automatic constructor which will enable an implementor to define new applications by means of a high-level system definition language. Based on such a definition, appropriate applications routines and driver tables will be generated.

REFERENCES

1. *Eighth Revision, International Classification of Disease Adapted for Use in the United States,* U. S. Department of Health, Education and Welfare, Public Health Service Publication No. 1693.
2. O'Kane, K. C., *The Design and Implementation of an Integrated Medical Information System,* Ph.D. Thesis, The Pennsylvania State University, 1972.
3. O'Kane, K. C. and R. J. Hildebrandt, *Operations Manual for the CSAR System,* Department of Computer Science, The Pennsylvania State University, 1973.

Interface for rapid data transfer and evaluation

by PRADEEP SHAH,* RUDY HAIDLE and GEORGE CZERLINSKI

Northwestern University
Chicago, Illinois

INTRODUCTION

Minicomputers are quite useful for the acquisition and reduction of data from physical and biomedical research equipment. Minicomputers have become progressively less expensive and are now comparatively easily available for research investigators. At that point, the problem arises to interface minicomputers effectively to research equipment. This interfacing task became recently somewhat facilitated by specialized books and articles.[1,2,3] Quite recently minicomputers were utilized for multiple ion detection in combined gas chromatography—mass spectrometry.[4,5] A computer centered instrument for simultaneous measurements of absorption and fluorescence was also described quite recently.[6] Finally, a computer system for the acquisition and analysis of temperature jump data was described.[7] Although a Biomation 802 transient recorder was used as fast analog-to-digital converter by the latter authors, they utilized a Digital Equipment Corporation PDP 11/20-8k and a teletypewriter ASR-33 as equipment connected to the output of the Biomation 802. In this paper we will describe the use of a Digital Equipment Corporation PDP 8/e-8k minicomputer as active processor between the Biomation 802 and a Hazeltine 2000 CRT terminal, which is also connected to a Hazeltine dual tape cassette.

Before the use of the minicomputer, we had produced data on paper tape in rather large quantities. The rolls of paper tape were then transferred to a large computer via an overnight carrier. Unfortunately, the large volume of long rolls of paper tape caused some operational problems at the main computer center. Furthermore, ten minutes were required for the production of paper tape with about 1,000 (8 bit) points per experiment from the Biomation 802. For our experiments, three minutes are required for thermal equilibration after a temperature jump and also for full attainment of a new equilibrium value after a pH jump. In other words, our Biomation 802 transient recorder is utilized on two types of chemical relaxation experiments: perturbation with the temperature jump apparatus and with the concentration jump apparatus (a pH-jump is a special case of a concentration jump).

The current arrangement allows us to conduct our experiments about ten times faster than before. We fill tape cassettes with data which are subsequently transferred through communication lines onto a large computer for thorough statistical analysis. This paper will describe the layout of the software and the details of the hardware configuration. No changes were made on the Biomation 802, allowing us to use a teletypewriter and the previously employed interface (Dijiscan Model B203) as back-up for this computer system. However, we have not needed to use the back-up system since the minicomputer was first brought into operation in June of 1973. The equipment is used rather extensively in two research projects on the mechanism of action of enzymes.[8,9]

METHODS

The experimental arrangement centered around a Digital Equipment Corporation PDP 8/e-8k minicomputer, interfaced both to a Hazeltine 2000 CRT terminal with dual tape cassette unit, and to a Biomation 802 transient recorder. The latter is connected either to one of our temperature jump instruments[10] or to our pH-jump apparatus.[8] The details are best described by several figures.

A schematic of the temperature jump apparatus with detection of transmission changes is shown in Figure 1. A similar instrument is also available for detection of fluorescent changes, containing a half reflecting mirror between light source and cell (containing the chemical system). The mirror transmits excitation light and reflects fluorescence, emitted back from the cell. The temperature jump instruments are equipped with special AC-coupling circuits to improve the detection of small changes behind large (fast) transients.[11] Three types of stopped flow instruments are available with the slowest one in considerable use: This is a pH-jump apparatus with detection of transmission changes and a time resolution of about 0.1 seconds.

The layout of the equipment around the PDP 8/e minicomputer is shown in Figure 2 without a 10 times wide band

* Current address: Equipment and Software Systems, Burroughs Corporation, Detroit, Michigan 48232.

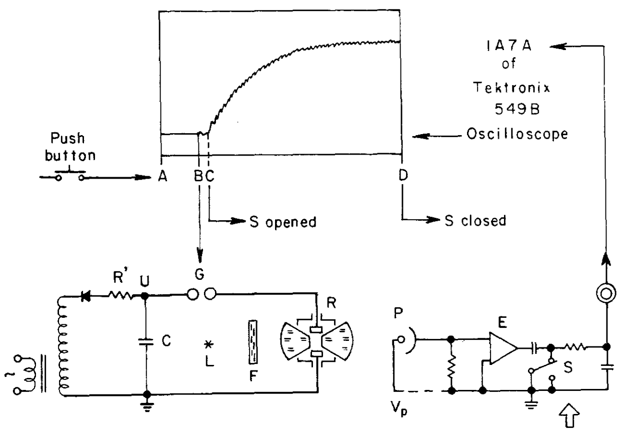


Figure 1—Schematics of a temperature jump apparatus with detection of transmission changes. The schematic shows abbreviated the current overall configuration of the apparatus. When a push button is pressed, a timing circuit is initiated, triggering first the oscilloscope deflection at A. After one centimeter of progression of the oscilloscope beam, a trigger pulse is generated at B, initiating the closing of the spark gap switch G. After about 0.5 millimeter further propagation of the oscilloscope beam, a trigger is generated at C, opening the grounding switch S in the detection circuit. The grounding switch is closed again at time D (or later). The extended grounding (from B to C) is used to prevent any processes to be shown on the oscilloscope screen, which are faster than the one currently under investigation and settings of the oscilloscope deflections, which are optimal for detection

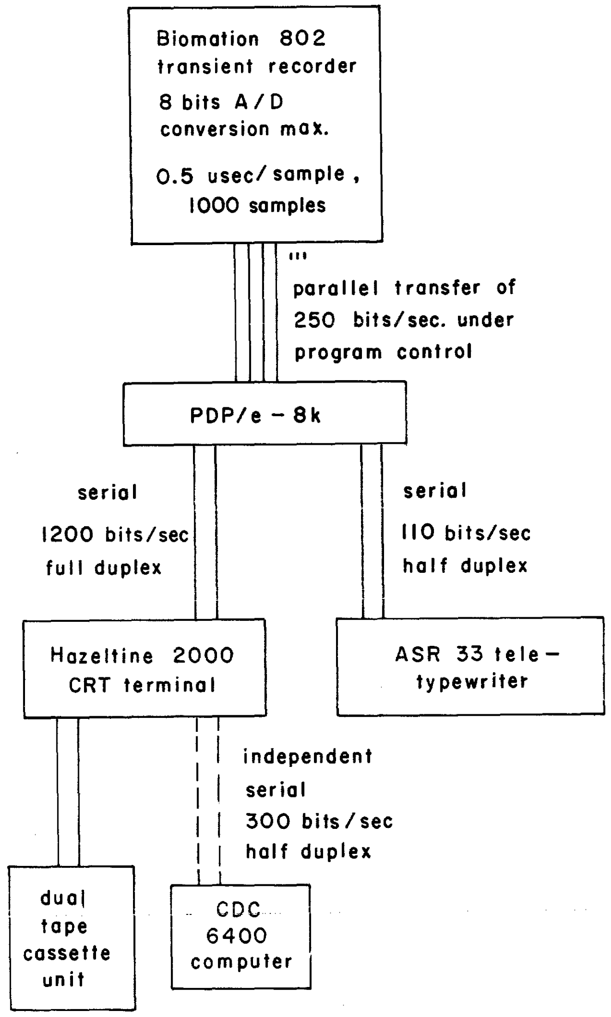
amplifier, which is located in front of the Biomation 802 and amplifies the signal derived from the temperature jump experiments. The data acquisition is controlled through interaction with displays on a Hazeltine 2000 CRT terminal. The controlling messages and the associated flow diagrams are shown in Figures 3, 4 and 5. More subroutines are actually used than shown in these three figures. A teletypewriter may also be connected and is essential for the initial loading of the binary program from a paper tape. (We encountered some problems in trying to use the Hazeltine cassette as alternate storage space for the programs: the problems are due to the fact that the interfacing Hazeltine terminal behaves inconsistently with respect to non-displayable characters, when transmitted for recording onto the cassette.)

The geometric layout of the various integrated circuits on the interface board of the PDP 8/e are shown in Figure 6. Enough space is left to add additional control functions. The functional circuit diagram is distributed over Figures 7, 8 and 9, all of which are mounted on one DEC interface board M1709. The remote control of the Biomation 802 is not implemented yet. The instruction set of the interface is as follows (where xx stands for 63, the device code for the transient recorder):

6xx0: Interrupt off.
Turns off the interrupt flip-flop, i.e., clears it. This stops any interrupts due to data ready signals from the Biomation 802.

6xx1: Skip on flag.
The instruction following 6xx1 will be skipped if the next word from the Biomation 802 is available and such a signal has already come from the Biomation 802 to the interface board.

6xx2: Interrupt on.
Turns on the interrupt flip-flop, i.e., sets it to 1 and this way enables the Biomation equipment to interrupt normal execution of a program when next word is ready on the line.



Biomation 802 transient recorder
8 bits A/D conversion max.
0.5 usec/sample, 1000 samples
parallel transfer of 250 bits/sec. under program control
PDP/e – 8k
serial 1200 bits/sec full duplex
serial 110 bits/sec half duplex
Hazeltine 2000 CRT terminal
ASR 33 tele-typewriter
independent serial 300 bits/sec half duplex
dual tape cassette unit
CDC 6400 computer

Figure 2—Overall diagram of the computerized data acquisition system. The output of the amplifier in Figure 1 is connected to an isolation amplifier, producing also a ten times amplification for adaptation to the input characteristics of the Biomation 802 transient recorder. A Tektronix 602 storage oscilloscope is connected to the Biomation 802 to show the stored information. The transient recorder is directly connected to a PDP 8/e minicomputer with control information normally appearing on the Hazeltine 2000 CRT screen. Some minor editing is available before the data are transferred from the core of the PDP 8/e onto the Hazeltine tape cassette unit. From there, data are transferred onto the disk of a large CDC 6400 via telephone lines

6xx3: Available for future expansion.

6xx4: Load the control register.

This instruction will output the control bits which may do one of the following:

(a) Effectively control execution of instruction 6xx5 (load)
(b) Effectively control execution of instruction 6xx6 (read)
(c) Start output from the Biomation 802. This will clear flag and pin 7 of the Biomation 802 will be grounded. (Ref: Biomation Manual—the user is required to ground pin 7 to start data transfer.)

Accumulator is cleared at the end of the instruction.

6xx5: Load scales or control lines.

Depending upon bit setting of instruction 6xx4, lower 6-7 bits of the accumulator are loaded into either the "voltage scale" buffer or "sweep time" buffer or "extra control lines"

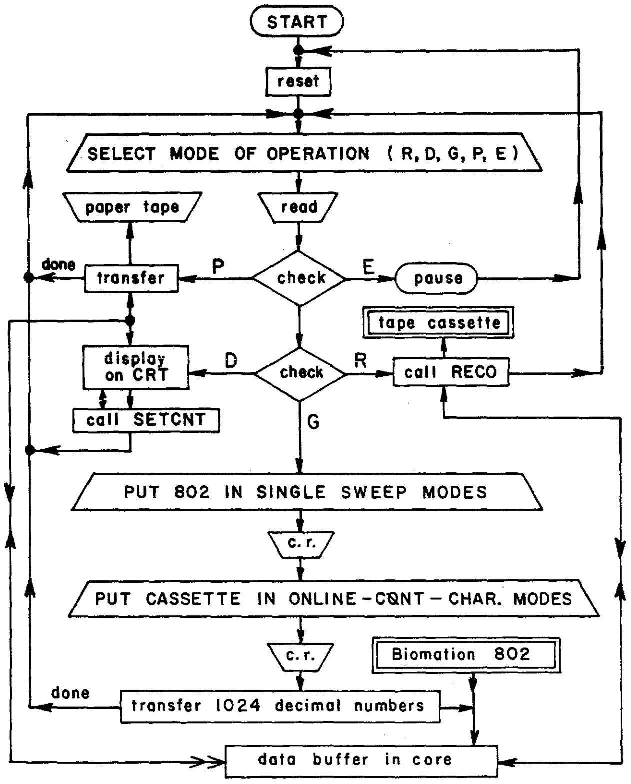


Figure 3—Program Flow Chart of main program in the minicomputer and of short subroutines. The actual text of the control information from the minicomputer is directly shown. The five letters in the first print statement stand for:

R = *R*ecord data in core onto dual tape cassette
D = *D*isplay data in core on the CRT terminal
G = *G*o to *G*et new data from Biomation 802
P = *P*unch data onto paper tape of tty
E = *E*nd the program.

Starting point is at the beginning of core of the minicomputer, address 0000. The main program starts at Octal 0200. The data buffer starts at Octal 2000 and is Octal 2000 long. Location of the messages follows and the program ends at Octal 4400

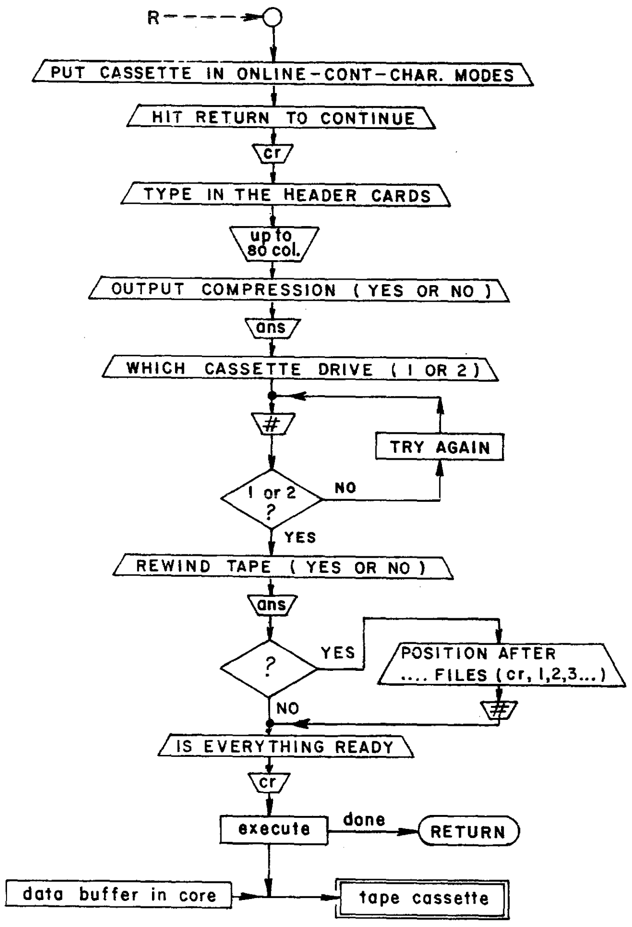


Figure 4—Flow Chart of subroutine SETCNT, responding to branch D. Sections of collected data may be presented on the screen without further editing capability

buffer register. Accumulator is cleared at the end of the instruction.

The "voltage scale" and "sweep time" buffers drive corresponding scale lines high only if "voltage scale" and "sweep time" knobs are positioned "external" on the front panel of the Biomation 802.

"Extra control lines" (6 of them) give open collector output which can be pulled up to +15 volts (maximum current of 20 ma). Their main use will be for "relay" controls. (Note: When "sweep time" multiplier is neither x1 nor x2 then it is x4. Similarly when "voltage scale" multiplier is neither x1 nor x2 then it is x5.)

6xx6: Read scales or control lines.

Depending upon instruction 6xx4 bit setting, "voltage scale", "sweep time" or "control lines" are *sensed* and loaded into the accumulator. If more than one "read" control is turned on by 6xx4, the effect will be to "OR" the lines into the accumulator.

Control lines are [compare sections 5.5 to 5.7 of Biomation Manual] (i) Z Enhance (pin 6) goes to 1, when "sweep

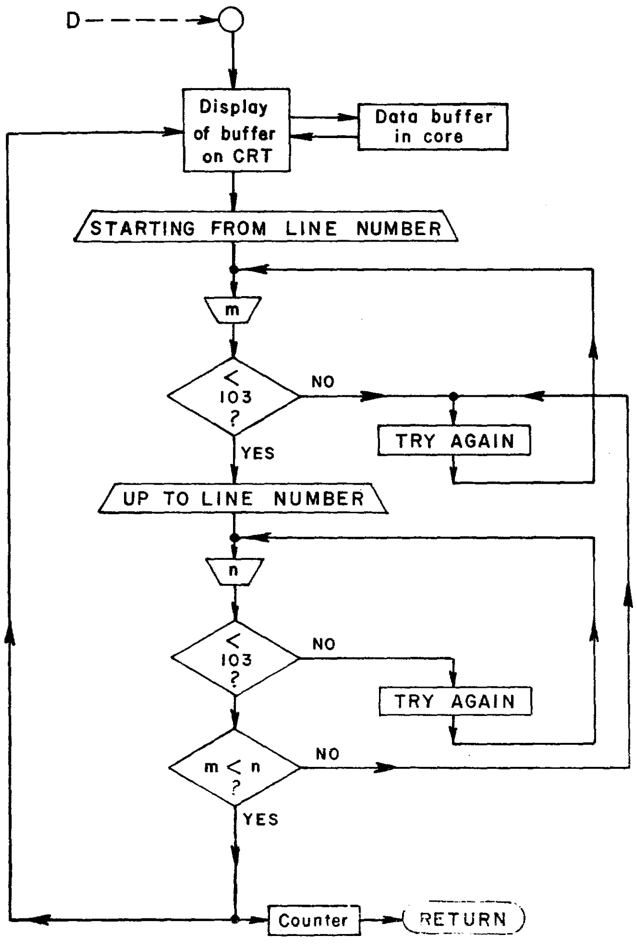


Figure 5—Flow Chart of subroutine RECO, responding to branch R. Before the data are transferred from the core into the tape cassette, everything has to be set so that subsequent processing takes place smoothly and the data are properly identified by the "header card", a line of up to 80 columns. Also some controls for the cassette are included in this program. The text as actually printed on the CRT screen is explicitly listed in the print statements. The printing "Position after..." represents an abbreviation of the question: "How many files would you like to skip forward?"

time" scale goes from A to B; (ii) External Sweep (pin 28) equals 1 if front panel time switch is set to "External"; (iii) External voltage scale (pin 26) equals 1 if front panel voltage scale knob is set to "External".

6xx7: Word command.

Read next word from Biomation into the accumulator, lower eight bits. This instruction will clear the "flag" and then the Biomation 802 will set it when next data is ready. If the entire Biomation 802 buffer is to be transferred into the PDP 8/e the following need to be done:

(i) Accumulator = 1; 6xx4, start output mode.
(ii) Execute 6xx7 1024 times to get 1024 points and store them.
(iii) Accumulator = 0; 6xx4 to return Biomation 802 to normal display mode. Note: Two ranges of time in which one can transfer data from the Biomation 802 into the PDP 8/e exist: (a) less than 500 μsec per word (b) greater than 2 msec per word.

If time taken to transfer data is not critical, range (b) is recommended.

RESULTS

Over the last few months, data were acquired with the described equipment on three different instruments, namely the temperature jump apparatus with detection of transmission changes (electron transfer experiments on cytochrome *c*), the temperature jump apparatus with fluorescence detection (binding of the fluorescing coenzyme NADH to liver alcohol dehydrogenase in the presence of inhibitors), and concentration jump experiments with detection of transmission changes at 633 millimicrons (detecting the various protonic forms of ferricytochrome *c*).

After users became familiarized with the various parts and their integrated action, they rapidly acquired mastery of the equipment and could perform soon experiments faster than before. Previously, the output of the Biomation 802 was transferred through an interface into a teletypewriter paper tape punch. As the teletypewriter produces only ten characters per second, ten minutes are required for the outputting of the data. With this new equipment, data are transferred out of the Biomation 802 and into the minicomputer within four seconds for a full set of 1024 "points". Although the subsequent transfer into the Hazeltine cassette unit takes a bit longer, a new experiment can in fact be conducted every three minutes, adequate for thermal equilibration in the thermostatted temperature jump cells and for almost all of the concentration jump experiments.

Although enough space is left on the PDP 8/e-8k, to write programs for the evaluation of the data, only the program

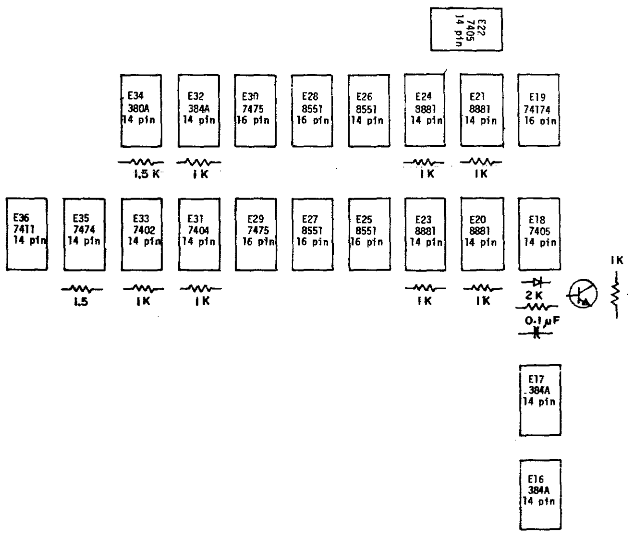
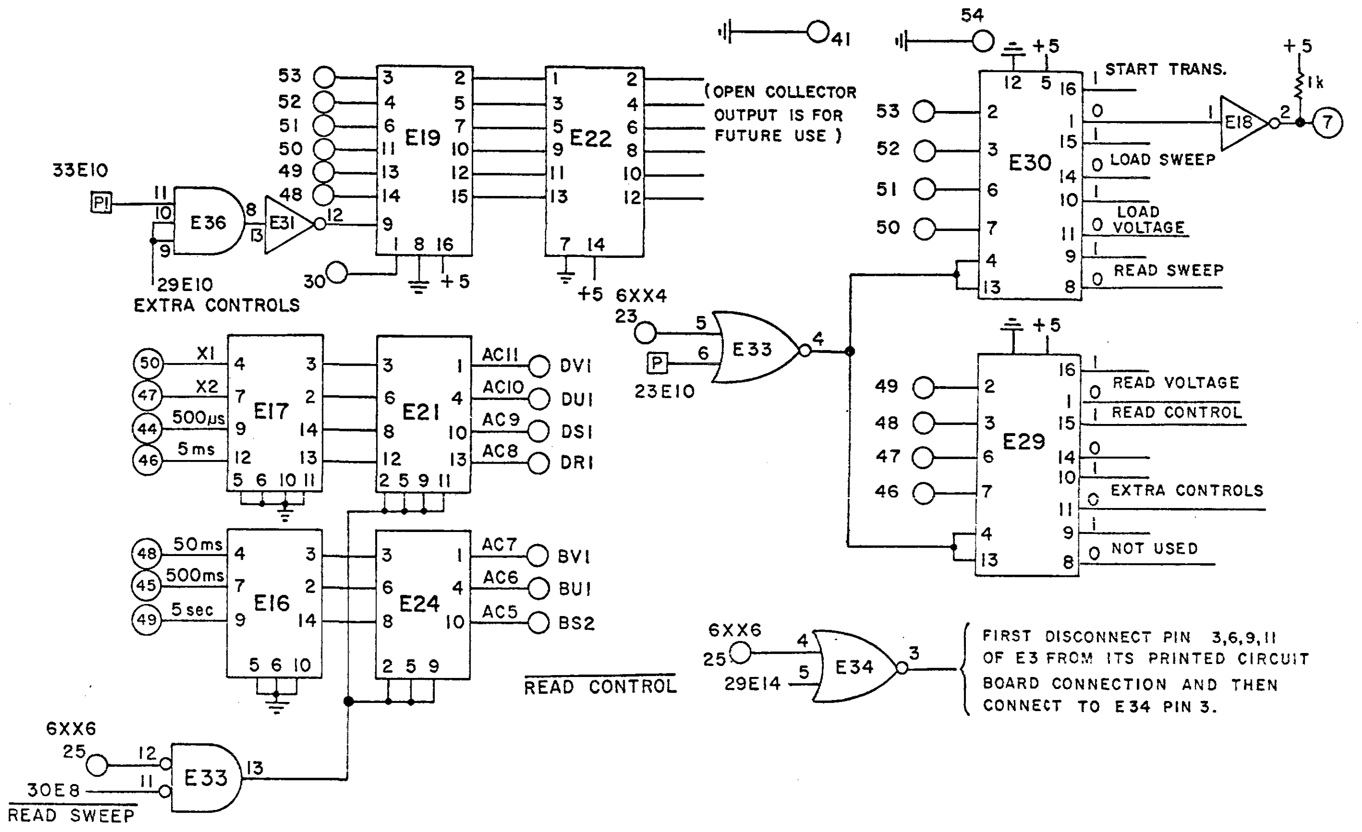


Figure 6—Layout of integrated circuits on the interface board of the PDP 8/e; encircled numbers refer to Biomation 802 pins



(OPEN COLLECTOR OUTPUT IS FOR FUTURE USE)
START TRANS.
LOAD SWEEP
LOAD VOLTAGE
READ SWEEP
READ VOLTAGE
READ CONTROL
EXTRA CONTROLS
NOT USED
READ CONTROL
FIRST DISCONNECT PIN 3,6,9,11 OF E3 FROM ITS PRINTED CIRCUIT BOARD CONNECTION AND THEN CONNECT TO E34 PIN 3.
READ SWEEP
EXTRA CONTROLS

Figure 7—Control circuit diagram for the interface board of the PDP 8/e

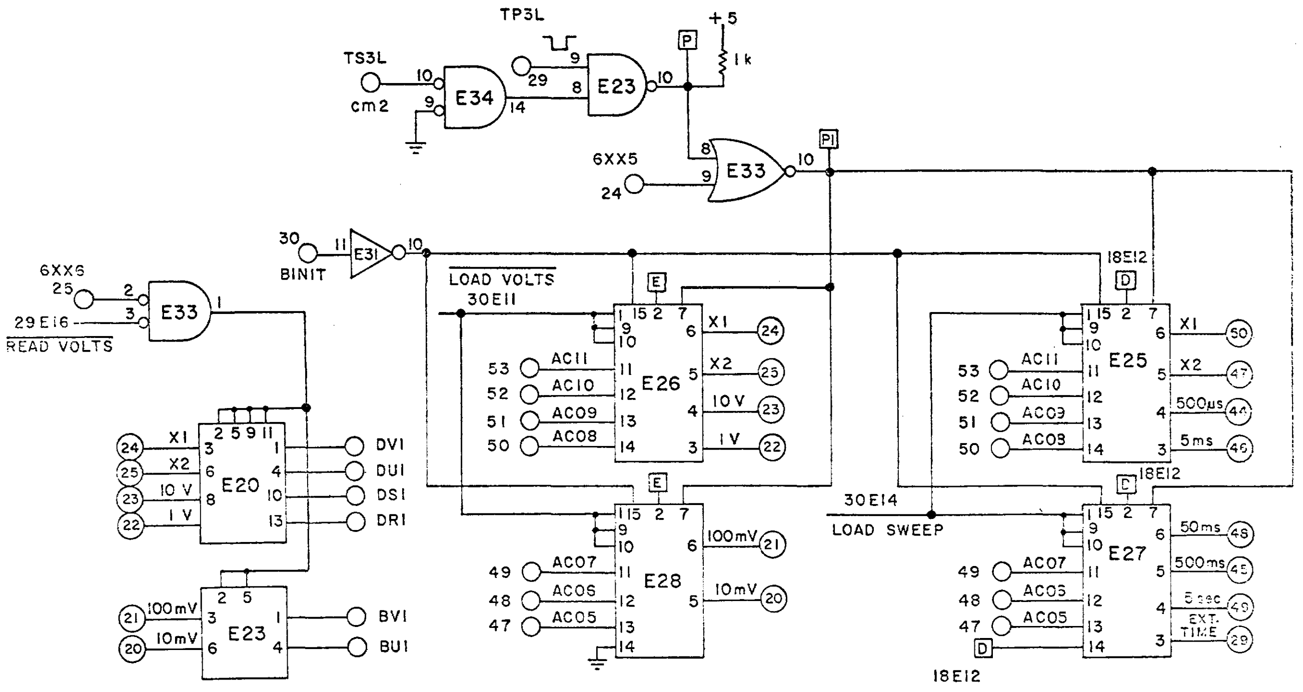


Figure 8—Load and read circuit diagram for the interface board of the PDP 8/e

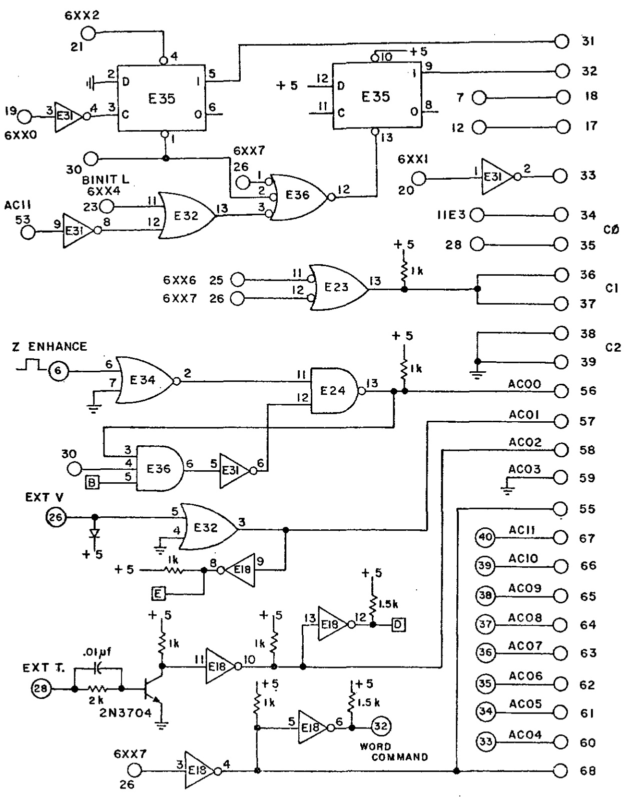


Figure 9—Gating circuit diagram for the interface board of the PDP 8/e

described in Figures 2, 3 and 4 was implemented. We decided on a small software package for the PDP-8/e, as a large library of data evaluation programs had previously been developed and was available in FORTRAN on the CDC 6400 of Northwestern University's Vogelback Computing Center. All programs are written for interactive processing, originally for use by teletypewriters. A program on the disk of the CDC 6400 provides for transfer of data from the Hazeltine dual tape cassette onto the disk of the CDC 6400 via ordinary communication lines. The data are then thoroughly evaluated in several stages, as schematically shown in Figure 10. Further details on the evaluation of data are presented in two Appendices. Appendix I describes the sequence of computer programs for the evaluation of data derived from pH-jump experiments (compare Reference 8), in Appendix II a sequence of computer programs is described for the evaluation of data from chemical relaxation experiments on a full enzyme catalyzed reaction cycle (compare Reference 9).

DISCUSSION

As was mentioned in the Introduction, an alternate computer based data acquisition and reduction system for chemical relaxation experiments is available. In that case, considerable software was developed and is kept in 8k of core of the minicomputer for further data reduction. This method

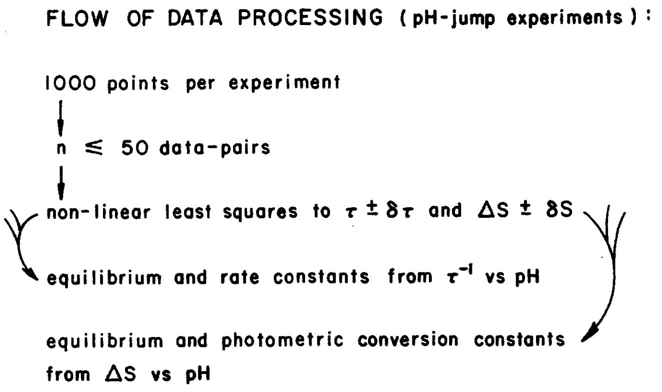

FLOW OF DATA PROCESSING (pH-jump experiments):
1000 points per experiment
n ≤ 50 data-pairs
non-linear least squares to τ ± δτ and ΔS ± δS
equilibrium and rate constants from τ⁻¹ vs pH
equilibrium and photometric conversion constants from ΔS vs pH

Figure 10—Flow of Data Evaluation, as used in a concentration jump apparatus (pH jump within about 0.1 seconds and subsequent changes much slower). Although a floating point package was available, it was not used because of its size and as a subsequent program ("TRANSL", see Appendices) was already available on disk and operating on integer data input. The data, as produced by the Biomation 802, are stored in numbers from 0 to 255

is quite feasible if punched paper tape is the only medium of permanent storage of data. As we have higher speed equipment available, we prefer to keep our data initially on dual tape cassettes, subsequently for on-line evaluation on disk (of the CDC 6400 in Evanston) and matrices of primary and secondary data on seven track IBM magnetic tape (at high density). IBM punched cards of these data matrices can also easily be produced. To facilitate identification of individual data sets, every set contains a header- and a trailer-card, containing system specific information. The software is designed such that data sets are easily sorted out and interpreted.

Although the described equipment serves mainly for rapid data acquisition, the operating program may be further expanded, and still a considerable amount of central core is available for programming. However, it is doubtful that any non-linear least squares analyses could be conducted within the PDP 8/e without the addition of a disk with support system. A current disadvantage is the slow transfer rate of the data into the large CDC 6400. An increase in the transmission rate to the CDC 6400 would be highly desirable. Nevertheless, the current configuration is of considerable advantage and should be of interest to others, utilizing transient recording equipment.

ACKNOWLEDGMENTS

The authors appreciate funding of the PDP 8/e minicomputer by the National Science Foundation (GM-20823) and of all associated work by the National Institutes of Health (AA00282-02). They are also indebted to Drs. M. Wagner and R. Zabinski for testing the usage of the equipment.

REFERENCES

1. Malmstedt and C. G. Enke, "Digital Electronics for Scientists," Benjamin, N.Y., 1969.

2. Enke, C. G., and R. E. Dessy, "Laboratory Computer Interfacing," Prelim. Edition, W. A. Benjamin, Inc., Menlo Park, Calif.
3. R. E. Dessy and J. A. Titus, "Computer Interfacing," *Analyt. Chem.* 45, 124A, 1973.
4. Holland, J. F., C. C. Sweeley, R. E. Thrush, R. E. Teets and M. A. Bieber, "On-Line Computer Controlled Multiple Ion Detection in Combined Gas Chromatography-Mass Spectrometry," *Analyt. Chem.* 45, 308, 1973.
5. Watson, J. T., D. R. Pelster, B. J. Sweetman, J. C. Frolich and J. A. Gates, "Display-Oriented Data-System for Multiple Ion Detection with Gas-Chromatography-Mass Spectrometry in Quantifying Biomedically Important Compounds," *Analyt. Chem.* 45, 2071, 1973.
6. Holland, J. F., R. E. Teets, and A. Timnick, "A Unique Computer Centered Instrument for Simultaneous Absorbance and Fluorescence Measurements," *Analyt. Chem.* 45, 145, 1973.
7. Hilborn, D. A., L. W. Harrison and G. G. Hammes, "An On-Line Computer System for the Acquisition and Analysis of Temperature Jump Data," *Computers and Biomedical Research* 6, 216, 1973.
8. Czerlinski, G., and V. Bracokova, "Kinetics of the Interconversions among the Electron-Transfer-Linked Forms of Ferricytochrome *c*," *Arch. Biochem. Biophys.* 147, 707, 1971.
9. Czerlinski, G., and J. O. Erickson, "Chemical Relaxation Studies on the Horse Liver Alcohol Dehydrogenase System," *Enzyme Communications*, I (in press).
10. Czerlinski, G., "Versatile Temperature Jump Apparatus for Following Chemical Relaxations," *Rev. Sci. Instr.* 33, 1184, 1962.
11. Czerlinski, G., "Timed Signal Grounding Switch for Observation of Low Level Signals Following Large Transients," *Rev. Sci. Instr.* 39, 1730, 1968.

APPENDIX I

Sequence of Computer Programs for the Evaluation of Data from the Cytochrome c-System

BIØI is a program stored on the Digital Equipment Corporation minicomputer PDP 8/e-8k and provides for the transfer of data from the Biomation 802 transient recorder to the Hazeltine dual tape cassette unit. Data sections may be displayed on the Hazeltine 2000 cathode ray tube and a heading to each set of data may be added from the key board.

PRETRAN is a program stored on disc of the Control Data Corporation 6400 computer in Evanston and called to transfer data from the Hazeltine dual tape cassette unit onto disc. Except for the control cards to obtain this program, all other systems operations are invisible to the user and built into the program via executive calls. The user only submits the name of the file, which he needs later on for further evaluation. This program also contains a subroutine to assist the user with difficulties, which he might have in obtaining data from the various instruments.

TRANSL is a program stored on disc of the CDC 6400 computer and called to reduce the data matrix to a size which can be easily handled by subsequent programs. As at least five data points from the Biomation 802 transient recorder are interdependent, due to the electronic rise time of the analog input circuitry, a minimum of five points ought to be averaged. Generally, ten to twenty points are averaged initially and the number of points averaged generally increases along the increasing data sequence at the option of the user. At this point, it is up to the user to distribute the points of the new data matrix reasonably well around the estimated relaxation time constants. If more numbers are supplied from disc than can be placed into a data matrix of fifty pairs, the remaining points are eliminated (this may occur when comparatively fast relaxation processes have been recorded on the Biomation 802; this should be generally avoided.) The abscissa of the data pair is computed from the time deflection and the number of data averaged. The ordinate is the average value of the data used for averaging and an error is also computed from the deviation of individual data points from the average.

DATAØPn is a program to analyze the input data matrix of not more than 50 pairs with a non-linear least squares subroutine, utilizing the Marquardt[1] method. A constant plus a sum of exponentials is generally assumed as underlying equation. For $n = 1$, the data may be evaluated assuming up to three exponential terms. The program with $n = 2$ is an abbreviated one, assuming only one exponential term and a constant. In most of our evaluations, we are able to utilize this abbreviated program. The program produces three parameters per data set together with a standard error per parameter. The parameter in the exponent is equivalent to an apparent rate constant (or better: the inverse of a chemical relaxation time) while the remaining two parameters correspond to signal amplitudes (or better: equilibrium signal changes). As many data matrices are processed by this program, a new data matrix is produced, containing these parameters as a function of analytical conditions.

CYTOC2 is a program to evaluate the factors in the exponents from the previous program as a function of pH, resulting in "true" rate parameters. This program utilizes a non-linear least squares analysis program and offers a variety of models for the interpretation of the data.

SEQUEN2 evaluates the signal amplitudes the (parameters obtained from DATAØPn) and according to models which correspond to protonic dissociations of protein forms. For pH jump experiments, no further evaluations are needed. However, for chemical relaxation experiments the protein concentration is varied initially at fixed pH. Another program (PØWERSE) is then used before the pH dependence of the apparent parameters is evaluated by CYTØC2 and SEQUEN2.

PØWERSE is a program utilizing a linear least squares analysis of a data matrix and useful for a variety of intermediate calculations. Standard errors of the parameters are also produced. Although the program allows for the utilization of a power series[2] as input function, we rarely find the need to analyze our data beyond linear terms (original functions are frequently rearranged to give a linear expression).

REFERENCES

1. Marquardt, D. W., "An Algorithm for Least Squares Estimation of Non-Linear Parameters," *J. of SIAM* 2, 431–441, 1963.
2. Golub in "Numerical Methods for Solving Linear Least Squares Problems," *Numerische Mathematik* 7, 206–216, 1965.

APPENDIX II

Sequence of Computer Programs for the Evaluation of Data from Liver Alcohol Dehydrogenase Experiments

Operation of the first four programs (namely BIØI, PRETRAN, TRANSL and DATAØPn) is identical to those, described in Appendix I. The algorithm of Marquardt[1] is thus used in DATAØPn. Output from DATAØPn is picked up by the following programs.

LADHn is a program specifically designed for liver alcohol dehydrogenase, with various versions available. Currently n=2 is in use and n=3 is in development. This program is used to analyze only the exponential parameters as a function of the analytical concentration of the components and of pH. No inhibitor is present. Equilibrium concentrations are computed from analytical concentrations with a specially developed subroutine[2] which solves the fourth order equation effectively by numerical means. Such a high order equation has to be solved, as the enzyme concentration is at least equal to or larger than the smallest dissocation constant and/or comparable with the analytical concentration of other components.

PØWERSE is an auxiliary program used for least squares analysis of data which can be described by a power series[3] (eventually after rearrangement). Generally, the power series is not extended beyond the linear term.

SEQUENn with n=3 or n=5, representing different versions. This program was designed for the analysis of experiments with liver alcohol dehydrogenase in the presence of imidazole. At this point, only the exponential factor is utilized in this program. A special subroutine[4] was developed, as in the absence of ethanol a cubic equation has to be solved to compute the equilibrium concentrations from the analytical concentrations. The subroutine computes these concentrations numerically in an effective manner. A variety of models are available to describe the experimental reciprocal relaxation times in terms of individual rate constants.

REFERENCES

1. Marquardt, D. W., "An Algorithm for Least-Squares Estimation of Non-Linear Parameters," *J. of SIAM* 2, 431–441, 1963.
2. Czerlinski, G. H., "Subroutine for Rapidly Converging Computations of Equilibrium Concentrations for Dehydrogenase Systems," *Comp. Prog. Biomed.* 1, 275–280, 1971.
3. Golub in "Numerical Methods for Solving Linear Least Squares Problems," *Numerische Mathematik* 7, 206–216, 1965.
4. Czerlinski, G. H., and R. Kobbe, "Subroutine for the Computation of Equilibrium Concentrations for the Elementary Biomolecular Cycle," *Progr. Biomed.* 3, 87–92, 1973.

An alternate interface to computers for the physically handicapped—the auto-monitoring communication board

by GREGG C. VANDERHEIDEN, ANDREW M. VOLK, and C. DANIEL GEISLER

University of Wisconsin-Madison
Madison, Wisconsin

INTRODUCTION

For many individuals severe physical handicaps have completely cut off most avenues of personal development and employment. Their physical involvement bars them from any constructive or creative activities requiring physical or manipulative abilities. Moreover, their inability to speak, write or efficiently operate even simple communication devices severely impairs their ability to develop and exercise their mental capacities. This latter problem is basically an output problem in which a normally functioning intellect is trapped within a body having no effective means of communicating or interacting with the environment. Fortunately with today's technology, especially micro electronics and the computer, new avenues are being opened for these individuals which promise them not only a chance for a more effective education and a more meaningful mode of self expression, but also a means of self support through employment.

The major problem in trying to realize the full potential of these individuals is in finding efficient means of communication for them. Information output should consist of both written communication and discrete commands with which they can control certain elements or devices in their environments. This paper will describe a new approach (The Auto-Monitoring Technique) and a new aid (The Auto-Com) which help to solve this problem for many severely physically handicapped people.

It will also describe how the computer can multiply the speed and effectiveness of this communication, further increasing the potential of these individuals. Implications for the education, employment, and overall enhancement of life for the physically handicapped will be discussed.

THE AUTO-MONITORING TECHNIQUE

The first problem in applying the potential of the computer to severely physically handicapped individuals involves providing an interface which they can control efficiently. There are basically three approaches that have been used to allow them some measure of control: encoding techniques, scanning techniques, and techniques employing a direct selection such as a keyboard.

The encoding systems utilize one or more switches which the person operates in a repetitive fashion to encode his output. The Morse code, for example, might be used in such an encoding system. This approach works best with people who have small but quick and well controlled movements such as might be found in the breath control of a para- or quadraplegic. For people with cerebral palsy and other afflictions which render them weak or limit their coordination, these types of aids are very slow and often cannot be operated without many mistakes.

The scanning systems are the most prevalent form of communication aid available today.[1-12] Unfortunately, few of them are computer compatible in their present form. In these devices, the alphabet is generally arranged in a rectangular matrix approximately seven by seven. The device steps an indicator or cursor across the columns until signalled by the individual using the device. The cursor then moves down the column until the individual signals the device again. The letter thus selected is then printed out on a typewriter, strip printer, or other output device. To control the scanning device, the person operates a single switch especially designed to take advantage of some one movement over which he has control. Because the scanning indicator must pause at each letter long enough for the user to activate the switch, this scanning process is very time consuming. This is especially true if the person has sporadic movements or cannot make a discrete response quickly. This approach does have the advantage that it can be used by almost any individual no matter how severe his physical disability. However, if the person can find some other means of control, it should be explored because of the very slow speed of this technique.

It was in trying to find an aid for those individuals whose movements were too sporadic or uncoordinated for the encoding systems and yet not restricted enough to resort to the scanning technique, that the auto-monitoring technique was discovered. The problem at hand was that of developing a communication aid for those individuals who have some gross pointing skills but who are unable to operate a key-

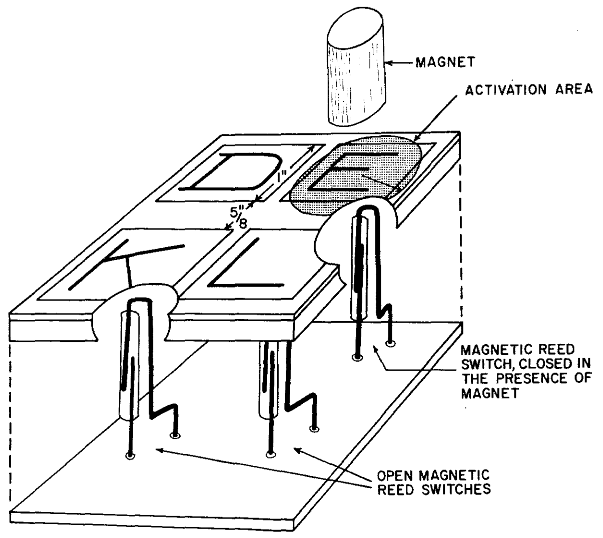

MAGNET
ACTIVATION AREA
1"
5/8"
MAGNETIC REED SWITCH, CLOSED IN THE PRESENCE OF MAGNET
OPEN MAGNETIC REED SWITCHES

Figure 1—Magnet (pointer) closes magnetic switch below surface whenever the center of the magnet is within the activation area. Switch closure is not acknowledged unless magnet remains within activation area for a fixed period of time

board of any kind, even with special modifications. A scheme whereby the user would directly select each output letter was deemed most appropriate for these individuals because of their ability to point and the inherent simplicity and speed of a direct selection technique. The problem then was one of finding a way to optimally utilize their limited pointing skills.

After many experiments, a solution was found to lie in the combination of a matrix of proximity switches located beneath a smooth surface and the use of a delayed activation mechanism. Using this technique, the operator need not push or pull any levers, buttons or switches. He need only slide a pointer into the vicinity of a switch and hold it near the switch for a short time (see Figure 1). In order for the system to respond, the particular switch involved must be closed continuously throughout the short, though adjustable, period of time. If the switch is opened before that period expires, the system resets, ignoring the switch closure. This feature means that switches only momentarily activated due to the passage of the pointer over them are ignored. Only when the pointer is kept within the sensitive area of a switch uninterruptedly for the set period of time, will the automonitoring system acknowledge the switch closure.

Thus the switch matrix is sensitive to lack of motion rather than to discrete motions as in normal switching arrangements. Pointing briefly to other switch locations as the user moves the pointer around over the surface will cause no false triggering. Nor will mistakes occur due to movements or momentary loss of control by the individual while trying to point to a given switch. If the operator loses control of his motion, the system just waits "patiently" until the operator regains his control. By adjusting the activation area of these proximity switches to the proper size, errors due to small tremors and inaccuracies of pointing were also almost entirely eliminated.

It should be noted that this delayed-action proximity-switching technique was modeled after the same process that a second person would use if he were to monitor a handicapped person's movements in trying to point to various letters painted on the surface of a board. Hence the term auto-monitoring.

Using this technique, it is possible to locate a large number of switches within an individual's range of motion without causing any problems due to accidental triggering of switches adjacent to the desired one. Moreover, because the proximity switches are located beneath the surface of the board, the operator need only slide the pointer around over the smooth surface of the board, never having to pick his hand up or support its weight in the air. The removal of the necessity to suspend his hand above the switching array and the removal of the vertical dimension from the required movement patterns have greatly increased the hand control of the cerebral palsied individuals worked with. The combination of all of these features has allowed even very severely cerebral palsied children to use devices which utilize the auto-monitoring technique.

THE AUTO-COM

The first application of the auto-monitoring technique is in the Auto-Monotoring Communication Board (Auto-Com). In the present model of the Auto-Com, the auto-monitoring technique is realized with a matrix of 84 magnetic reed switches mounted on 1⅜ inch centers. Each switch is located directly underneath the center of a letter painted on the surface of the board and can be activated by a magnet anywhere within a ⅝ inch radius of this center (Figure 1). Both visual and auditory feedback have been provided to aid the person using the board.

The Auto-Com itself is completely contained within a 18×24×1½ inch wooden case (Figure 2). It weighs about 8½ pounds and is designed to mount directly onto the arms of a wheelchair in the same manner as a lapboard would. In fact, the Auto-Com has been designed so that it can be used as a lapboard when not in use as a communication device.

In addition to the Auto-Com itself, there are two other necessary components of the system—a magnet and an output device. The magnet is mounted on whatever object or part of the body the child can best point with. In most cases the magnet is mounted on a handpiece consisting of a clear plexiglass base with a custom molded hand-grip (see Figure 3). The magnet, which functions as the pointer for the child, is mounted well away from the hand-grip. This affords maximum visibility of the letters as the child slides the magnet over them. To facilitate the sliding motion, the bottom of the handpiece is partially covered with felt. This handpiece serves both to stabilize the operator's hand and to smooth and damp his motions.

The principal output form of the Auto-Com at the present time is an ordinary television set equipped with a commercially available TV controller that allows the user to print letters on the television screen. This particular output

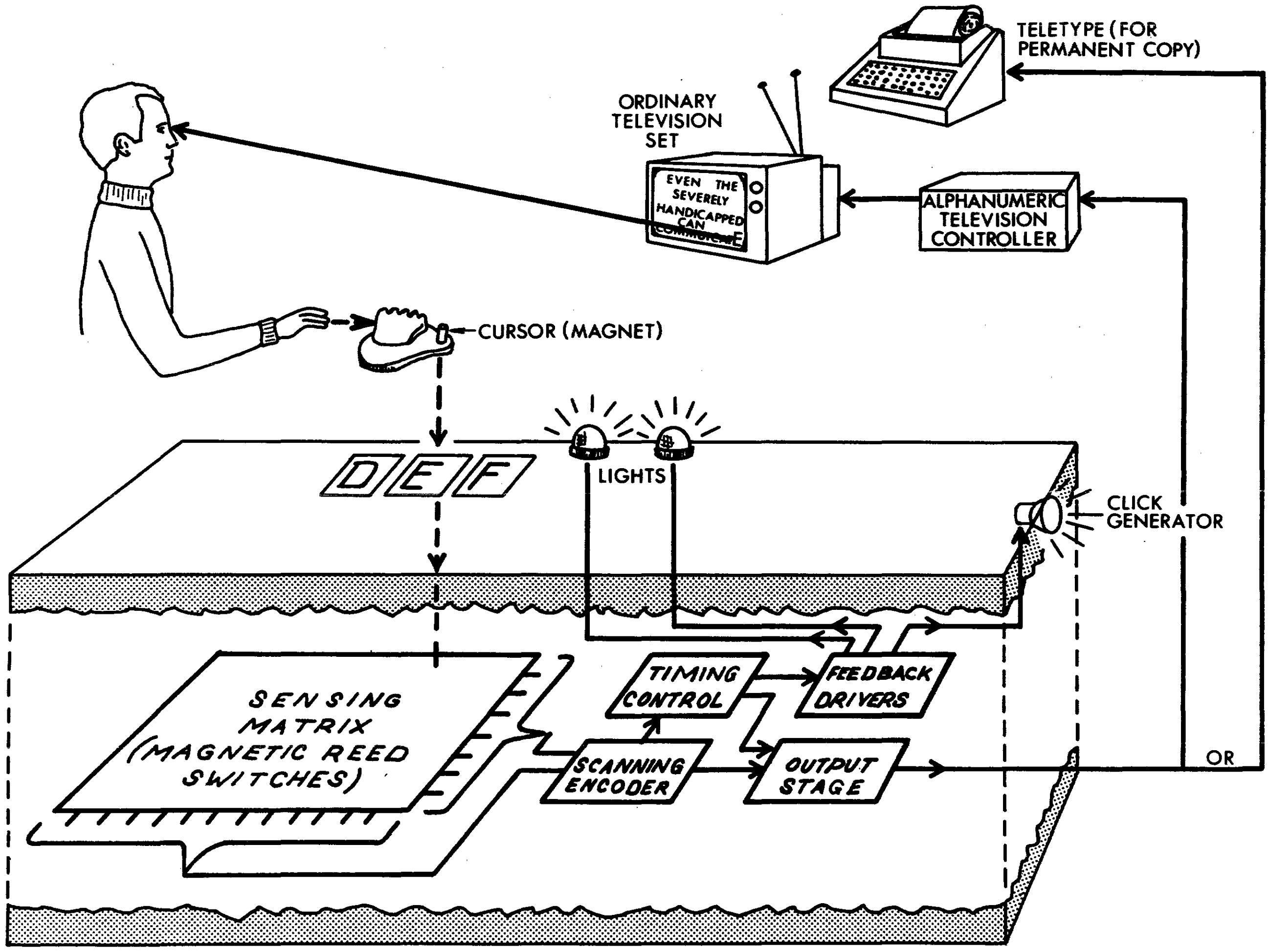

TELETYPE (FOR PERMANENT COPY)
ORDINARY TELEVISION SET
EVEN THE SEVERELY HANDICAPPED CAN COMMUNICATE
ALPHANUMERIC TELEVISION CONTROLLER
CURSOR (MAGNET)
D E F
LIGHTS
CLICK GENERATOR
SENSING MATRIX (MAGNETIC REED SWITCHES)
TIMING CONTROL
FEEDBACK DRIVERS
SCANNING ENCODER
OUTPUT STAGE
OR

Figure 2—Exploded diagramatic view of the Auto-Com system showing major functional blocks

has been chosen because it provides good visibility, feedback, portability, and correctability. The Auto-Com has also been used with other output devices: teletypewriters, special typewriters, and strip printers. Any output device using the standard ASCII Code can be used with the Auto-Com; the exact output form of which is a 20 milliamp current loop with the information in standard 110-BAUD 11-bit serial format. Thus the Auto-Com can interface directly with any computer accepting this standard serial input. Figure 3 shows a photograph of the Auto-Com with two of its output devices.

Operation of the Auto-Com is simple and straightforward. Even children can operate the Auto-Com with only minutes of instruction. To print a letter the user simply grasps the handpiece (a velcro strap arrangement is provided for individuals who have no grasping abilities) and slides the magnet over to the letter he wants printed. For those who cannot use the handpiece, the magnet can be mounted on a ring, headstick, shoe, or anywhere else that would be advantageous for him.

This simplicity of operation, combined with the two-dimensional movement feature of the auto-monitoring technique, has proven so desirable that several centers have indicated a desire to secure Auto-Coms to use with some of their students who now can use specially modified typewriters. The reasons they cite for wanting an Auto-Com despite its higher cost are ease of operation, reduction of mistakes, and increased use time before fatigue.

Future developments

The present program for the Auto-Com is centered around its development as a communication aid. A large emphasis is being placed on making it as flexible as possible and increasing its utility in the educational setting. Toward these ends, emphasis is being placed on the development of these features:

Complete portability

A portable model of the Auto-Com, which runs on batteries and contains its own miniature strip printer, has been designed. With this unit an individual will be able to move

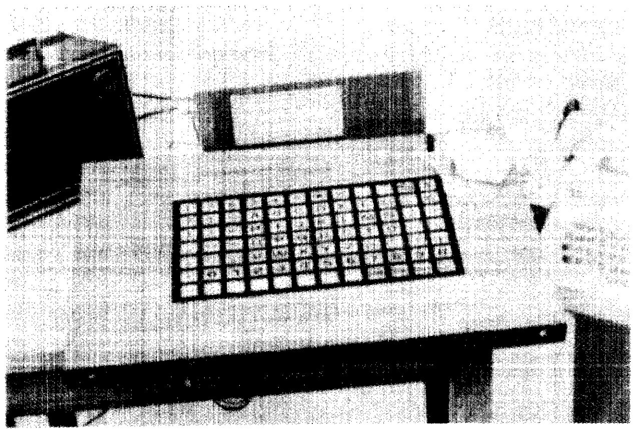


Figure 3—Earlier model of Auto-Com with handpiece and two of its output devices: a teletype terminal and a TV screen with Ann Arbor TV controller. A newer model will also have a built-in strip printer for output

freely about his school or home and always have his "voice" with him. Attached to his chair and doubling as a lap board, it will always be with him requiring no special set up for each use.

In addition to the strip printer, this portable Auto-Com contains a small FM transmitter which will enable it to control the TV controller without hookup wires. A large screen television set with TV controller could then be placed nearby and controlled by the individual from anywhere in the room. This feature is seen as particularly powerful in a classroom setting where the student could move about freely and participate in class discussions much like the other students by using the TV screen printout as his voice. The selection of either the strip printer or the serial ASCII telemetry unit is controlled by the user. In this manner the operator, when working in a computer environment, could easily switch back and forth between the printer (for communication) and the telemetry unit (for communication with a computer). He would also be able to talk to the computer from any position in the room, thus lessening his mobility problems.

Printed copy

Printed copy may be obtained at any time by simply connecting the Auto-Com to a teletypewriter or modified typewriter instead of the TV controller. However, this eliminates the feedback and correctability features of the television set. To provide for both printed copy and correctability a transfer option is being developed which will automatically transfer the contents of the television screen to a printing device on command from the Auto-Com surface.

Price

Throughout the design, heavy emphasis has been placed upon keeping the price of this aid to an absolute minimum. The rapid advance of integrated circuit technology has made this type of aid possible and economically feasible. As the technology increases, the cost will continue to decline. A major component of the Auto-Com cost still lies in the output device. For this reason, a large portion of our research effort has been directed toward securing or developing inexpensive output modes. If the Auto-Com were used in conjunction with a computer system, where it would be used mostly as an "alternate keyboard," the cost of the output system would be eliminated and the price further reduced.

An Auto-Com system, then, can take two basic forms depending on its intended use. It can take the form of a complete system if it is going to be used as a communication aid, or it can take the form of a simple keyboard if it is to be used as an interface with a computer or other data processing system.

When used as a communication aid, the final Auto-Com system will consist of two parts, each designed to handle different functions in communication.

(1) *The portable Auto-Com.* Highly mobile, this unit has a miniature self-contained, strip printer for its output. When used alone, it is designed primarily for conversation. With the FM output, it can also control a stationary output system.
(2) *The stationary output system.* These television and output printer media provide the feedback and page format most useful in educational settings and for extended independent work. The system will be designed so that several individuals, each with his own portable Auto-Com, could share a common printer or CRT display.

When used as a keyboard for data entry, the Auto-Com can take either the form of the full portable Auto-Com as described above, or a simpler form of the Auto-Com having no strip printer and deriving its power from the computer or a separate power source.

The Auto-Com wordmaster

A natural next step in the evolution of the Auto-Com as a communication aid is the addition of entire words to the surface of the board. With this option, the operator could then cause entire words, phrases, or sentences to be printed out by pointing to a single square on the Auto-Com. This would permit him to communicate in a word-by-word fashion as in speech rather than having to spell everything out. For computer use, commonly used words, phrases, or symbol sets could be substituted for the words.

An attachment dubbed the "Wordmaster" has been developed to do this. Now under initial evaluation, the Wordmaster/Auto-Com is expected to provide an increase in speed of approximately 2-4 times over that of the original Auto-Com. Moreover, it will enable even more people, especially children, to use the Auto-Com. The ability to spell will no longer be a prerequisite for use of the Auto-Com. It also opens up the possibility of using pictures on the surface of the board to specify the output words. This technique may be used either to accelerate reading skills or to provide communication for those who are prereaders or who have reading problems, but do possess adequate expressive skills.

THE USE OF COMPUTERS WITH THE AUTO-COM AND OTHER COMMUNICATION AIDS

There are two basic ways in which the computer and communication aids can be combined to augment each other. The first is an arrangement in which the communication aid serves as an "alternate keyboard," allowing handicapped individuals with no previous means of access to a computer to benefit from the computer's capabilities. In the second relationship, a computer is used to augment the communication aid by increasing the speed, efficiency, and utility of the aid.

As an alternate keyboard

When used as an alternate keyboard, the purpose of the communication aid is to provide the user with an interface that is specifically designed for his particular abilities. The simplest form of communication aid which can accomplish this purpose is the "guarded" keyboard. IBM makes special keyboard handguards and armrests for most of their typewriter models including their I/O typewriters. The Cerebral Palsy Communication Group has also developed a special keyboard guard for the Teletype Model 33 teleprinter. Using this keyguard, a cerebral palsied student at the University of Wisconsin-Madison has successfully completed the requirements for a B.A. Degree in Computer Sciences. For programming, he operated the modified teletype terminal from his dormitory room, using the time-sharing facilities of the University's Univac 1108 computer. In this manner, he was also able to overcome the transportation difficulties associated with constant travel to and from the Computing Center.

For individuals too severely handicapped to use any of these guarded keyboards, the Auto-Com may provide a solution. Because it has been designed to output in 110-BAUD serial ASCII, the Auto-Com is directly compatible with any computer which accepts input from a teletypewriter. Although the scanning and encoding aids available today are not directly compatible with computer inputs, they too could be modified to permit them to be used as computer interfaces for the handicapped. Thus, any person, no matter how serious his physical handicap, or what form it takes, can be outfitted with an aid which would allow him to readily communicate with a computer.

The relative unimportance of speed in working with a computer, either a dedicated mini-computer or a larger time-shared computer, opens up the possibility of computer-related jobs for the physically handicapped. The computer does not get impatient and the user is unhurried—he is able to work at his most comfortable and efficient rate. Moreover, the computer also offers the handicapped person the opportunity to remain at home while working. Phone lines provide him with a direct link between his home and the computer. Since mobility, and sometimes nursing care, are major problems for the physically handicapped, the ability to work directly from his home or from a rehabilitation center, can make the difference between a person being employable or not.

Access to computers also opens up to the physically handicapped the whole world of individualized instruction. In the education of the handicapped, where individualized instruction is very often needed but rarely available, the computers can provide the educational programs these students need at a pace in keeping with their physical abilities.

The power of the computer in individualized instruction is well illustrated by the PLATO project at the University of Illinois, Champaign-Urbana. This computer-based educational system is designed to handle up to a thousand remote terminals all communicating with a central computer. The system has educational programs ranging all the way from pre-kindergarten through post-doctoral studies. The PLATO terminal uses a standard computer keyboard as one of its input forms. These keyboards could easily be replaced by communication aids specifically designed for use by the handicapped, thus allowing them access to the PLATO system. The Auto-Com, for example, can be interfaced with the PLATO system by simply using a patch-cord.

To expand the function of a communication aid

When used in conjunction with a computer, the power of a communication aid can be greatly expanded. With limited ability to produce output signals, the handicapped person's rate of communication can only be increased by increasing the information sent by each of his commands. The use of computers, with their immense information storage capabilities and flexible peripheral devices can greatly increase the information transfer rate of the handicapped.

The Auto-Com with Wordmaster attachment, for example, presents the user with a choice of the alphabet, numbers, and a list of 191 words, a number limited largely by memory storage and display considerations. Sixty-three of the words have been predetermined and are stored in a permanent memory. They are also printed on the board and are accessible through an "upper case" arrangement. The other 128 words are chosen by the individual user and are written into an interchangeable ROM unit. These latter words are printed on interchangeable cards and can be selected in "third" and "fourth case" modes. Using a computer, this vocabulary could be greatly expanded and its display and selection features considerably improved. A large enough wordset could be established so that the user could converse easily in a completely word-by-word fashion instead of having to spell out most of his messages letter-by-letter. Even phrases and whole sentences could easily be stored in the computer, increasing the options of the user still further. At some point the vocabulary size will get so large that it would take more information to specify a specific entry than it would take to spell it out. Work is now being done to identify that boundary and to study ways of developing vocabularies that can be made fully functional without reaching that point of inefficiency.

The use of computers in the communication problem of the handicapped need not be limited to maximizing output information only. The text storage and editing capabilities available in many present computer terminals could prove of immense value to those physically handicapped who cannot otherwise easily review, correct, or modify their written work. The written work of others could also be stored and transmitted in digital form to the handicapped person in his home or work area. It is not inconceivable that whole books, magazines and other written information could be made available to the physically handicapped in this form (both for recreational and professional use). Answering the phone (perhaps with computer generated speech), controlling a room's ventilation and lighting, and providing a good measure of self-care are all feasible today through computer controlled devices.

It has been suggested by futurists that within a few decades many workers in the U.S. will be able to perform their jobs from their homes using communication technology now being developed. The video-telphoene, two-way cable TV, and computer technology all suggest more flexibility for home-centered employment. The ability of handicapped individuals to use these devices will enable them to take a more useful and rewarding place in that future society.

In other applications, the computer could contain algorithms for controlling machines with complex functions. In this useage, the handicapped person would specify the various operations he wanted performed and the computers would execute the various steps necessary to perform them. One command from the handicapped person could initiate a string of individual commands from the computer, thus increasing the effective speed and efficiency of the handicapped person.

Thus the combination of the computer and communication aids can provide the physically handicapped with many opportunities not otherwise available to them. These opportunities touch many areas: education, personal advancement, employment, recreation, communication, and social interaction. Furthermore, the cost of these systems should continue to decline with the advance of technology and the increase in the development of low cost electronic functional modules. If the concept of home based employment via tele-communication links proves economically and administratively feasible, then the occupational opportunities for the physically handicapped seem to be limited only by the efficiency of the communication and interaction systems that are available. As new and more efficient techniques for utilizing the intact abilities of the handicapped are developed and interfaced with computer systems, the effective capabilities of these individuals will continue to increase allowing their mental capacities to be more and more fully realized and utilized.

ACKNOWLEDGMENTS

We would like to thank all those members of the Cerebral Palsy Communication Group who have contributed to the development of the Auto-Com/Wordmaster. In particular, we would like to note the efforts of David F. Lamers, Warren P. Brown, Gerald A. Raitzer, Robert J. Norton, David E. Church, Deberah Harris, and Claudia L. Scheibel. The technical counsel of Professor Leo Jedynak is also acknowledged.

We would also like to thank all those who have provided the monetary support necessary to carry out this research, especially the University of Wisconsin, the Madison Public School System, the Robert J. Ritger Memorial Fund, the Dane County Chapter of United Cerebral Palsy, the Bacon Foundation, and most recently from the National Science Foundation, who is presently funding our research efforts.

Many industrial concerns have also contributed to the group in the form of information and the donation of products. The most notable of these are Ann Arbor Terminals, Ann Arbor, Michigan; Hamlin Inc., Lake Mills, Wisconsin; Texas Instruments; Intel; Signetics; and Canon Inc.

REFERENCES

1. Stalder, E. K., Editor, "Patient Uses Ingenious Device As Means of Communication," *The NIH Record*, Vol. XVI, No. 24, p. 1.
2. Excerpts from Annual Report—Orthotic Research Unit, Ontario Cripple Children's Centre, 350 Rumsey Road, Toronto, Ontario, 1972.
3. Sampson, D., "A Communication Device for Patients Unable to Speak," *Med. and Biol. Engrg.*, Vol. k, pp. 99-101.
4. Hume, B. C., General Manager, *Centre Industries*, Allambie Road, Allambie Heights, New South Wales.
5. Hackler, N., *North Electric*. Galion, Ohio (Personal communication).
6. King, G. (Editor), "Communication System for the Handicapped," *Electromechanical Design*, Vol. 17, No. 1, p. 6.
7. Howard, W., *Bush Electric*, 1245 Folsom Street, San Francisco, California 94103.
8. Jefcoat, R., P.O.S.M. Research Project, 63 Mandeville Road, Aylesbury, Bucks.
9. Steele, J. D., *Zambette Electronics Ltd.*, 3, Avon Way, Shoeburyness, Essex 5s39DZ.
10. Kafafian, H., *CRI Second Report*, Cybernetics Research Institute, 2233 Wisconsin Avenue NW, Washington, D.C. 20007 (Personal conversation).
11. MEFA GmbH Bonn, 518 Eschweiler, Postfach 466, Germany.
12. Micheelsen, V. Wissing, Reva Aids, Solvgade 32, Copenhagen K, Denmark.

A computing environment for the blind

by MORTEZA AMIR RAHIMI and JOHN B. EULENBERG

Michigan State University
East Lansing, Michigan

INTRODUCTION

Much of the early work on applying computer technology to the development of sensory aids for the blind was devoted to systems for transforming ink-print texts into a Braille format.[2,6,12] This continues to be an important use of computers, but the disadvantages of Braille render it a poor substitute for the modes of information access normally available to people with normal sight. Braille is embossed on bulky paper. The surface area required for tactile discrimination of the Braille dots add to the problem, since this means that a Braille text takes up more space than a corresponding ink-print text. Furthermore, the ability to read Braille takes considerable time to acquire, and not all visually handicapped persons can read Braille, due to concomitant handicaps or because their visual impairment came fairly late in life. Braille also has the disadvantage that it is unreadable to almost all sighted persons; blind persons cannot read what sighted persons can read, and vice versa. Many blind persons are excellent typists, so they can indeed communicate in writing to the sighted, but they have no way of proofreading their own typing.

All of these facts call for alternate channels of communication for the blind. Nowhere is this more keenly felt than among those blind persons who are involved in computers and who are sensitive to the great potential which computers hold for information processing and for transforming one mode of representation into another. A recent issue of the Newsletter of the ACM Special Interest Group on Computers and the Physically Handicapped [SIGCAPH NEWSLETTER, Number 8, July 1, 1973] carried an article summarizing the responses of blind computer programmers to a comprehensive questionnaire on their experiences and special needs. Among the responses to the question "What special tools or equipment, if any, would you like to see developed?", were these: "faster means of reading than braille", "a machine to read inkprint printouts from the computer", "efficient method of verifying punch cards", "card reader enabling me to make corrections myself", "faster way of inserting cards in the card deck", "device to show which columns I am punching", "auditory output or input echo", and "random access books".

One means for meeting these demands is by adapting a digitally controlled voice synthesizer to provide understandable computer output through the auditory channel.[1,3,4,5,7,8,10,11] This application of electronic voice synthesis has been apparent to workers in the field of voice and speech synthesis for a long time, especially in the form of systems for going directly from ink-print to speech, incorporating optical character readers.[9] Unfortunately, the actual machinery for delivering voice output has been large and expensive, usually taking the form of one-of-a-kind monstrosities in research laboratories. Even when made commercially available, they required considerable sophistication in acoustic phonetics and electronics on the part of the user.[4,5]

One alternative to the speech synthesis approach is the vocal response unit, which allows fast random access to encoded audio recordings of actual human speech. A repetoire of words and short utterances can be stored on a computer memory, and a computer can rapidly assemble concatenations of these speech units to form intelligible simulated spoken output. The principal disadvantages of this method lie in the relatively large amounts of computer power which must be devoted to storing and accessing the speech data and in the substantial initial investment required in making the vocal response unit available.

The recent advent of relatively inexpensive, easily portable, digitally controlled electronic voice synthesizers which can convert signals representing phonetic transcription into the corresponding acoustic signal has made practical the development of voice-mode computer communication stations which make only modest demands on the computer.[11] This means that the considerable information storage and information manipulation capabilities of the computer can be exploited by a blind user without the intermediary of Braille. Such functions as document preparation and text editing, information retrieval, and interactive mathematical calculation can now be implemented in an output format which does not involve the printed page. Of course, this use of the auditory channel for computer output can also be used by sighted users, and, in fact, this constitutes an advantage over the use of Braille output, since it is a mode of output which can be shared by both the blind and the sighted at the same time.

In this paper, we will describe our work in developing just such a communications station. We will briefly describe the hardware and software needed to implement the station,

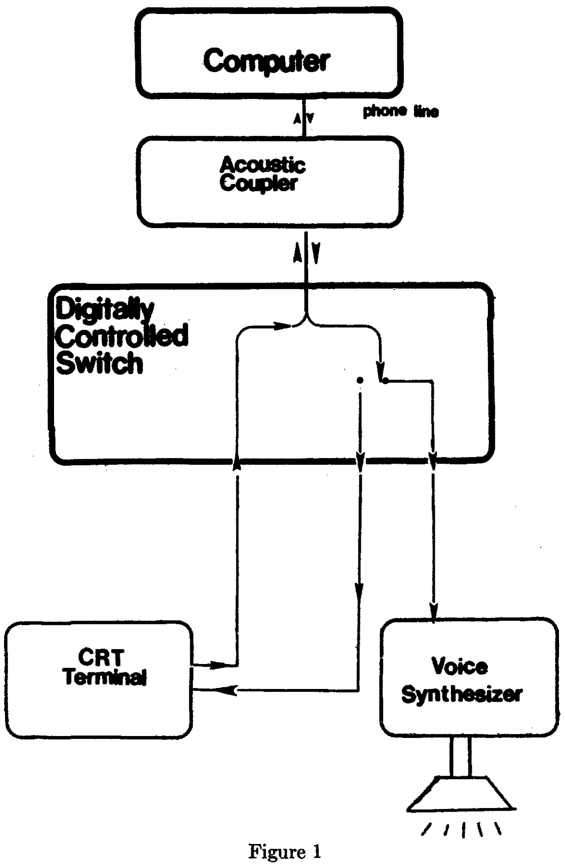


Figure 1

and we will discuss the actual experimental use of our system by the blind. Finally, we will give our views on the implications of our work for the development of new types of computer languages designed to take advantage of the auditory mode of output.

Hardware and software

The design of software and hardware was guided by the following set of objectives:

- The cost of the terminal should be comparable to that of ordinary CRT or teleprinter terminals.
- The terminal should use standard communications equipment.
- The terminal should be usable with any timesharing interactive system.
- The software involved should be implementable under all standard operating systems, requiring no special modification to the system.

Hardware

Figure 1 shows the hardware configuration. Is consists of a voice synthesizer, a speaker, a standard terminal keyboard, and support electronics. Communication with the computer is achieved normally at the rate of 300 Baud which supports the speech rate of the synthesizer. This Baud rate is not essential since the terminal does have a buffer memory.

The phonetic images of words and sentences are built up by concatenation of 8-bit symbols representing the phones of English in the buffer memory. The string in the buffer is sent to the synthesizer for pronunciation when a special control character is received by the buffer mechanism. The 8-bit code consists of six bits representing the phone and two bits representing one of four levels of intonation. Since the communication takes place with 7-bit ASCII characters, the 4 lower bits of two consecutive ASCII characters are actually used.

The terminal allows for echoing of the keys without access to the computer.

Software

Two types of software are available. First, the software designed specially for voice output. This includes a text editor, a Fortran interpreter, and a number of other programs. In these programs the output messages are phonetically encoded for direct transmission to the terminal. The second type of software includes routines to convert the output of any program to the desired phonetic form. Using a lexicon as well as the rules of orthographic-to-phonetic correspondence, this conversion package will change any numeric or English text to phonetic transcription. Components of this package and their interconnection are shown in Figure 2.

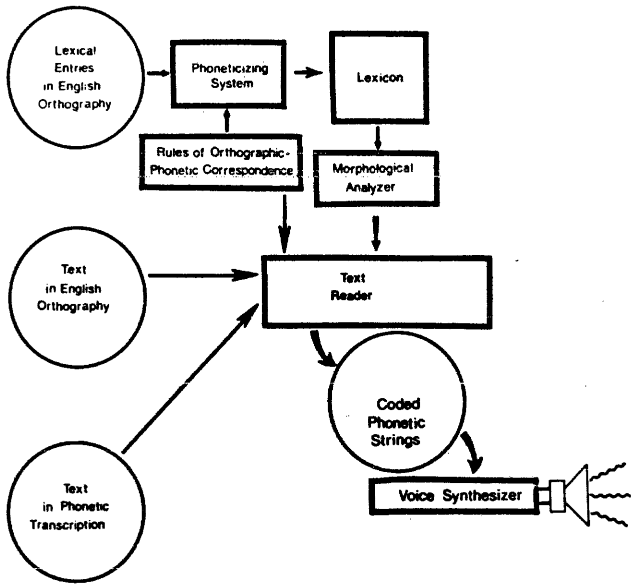

Lexical Entries in English Orthography
Phoneticizing System
Lexicon
Rules of Orthographic-Phonetic Correspondence
Morphological Analyzer
Text in English Orthography
Text Reader
Coded Phonetic Strings
Text in Phonetic Transcription
Voice Synthesizer

Figure 2

APPLICATIONS OF THE EXPERIMENTAL COMMUNICATIONS STATION

We have as yet had only a limited amount of experience with actual use of our system by the visually handicapped. For the most part, the response has been enthusiastic, and we have received quite a few suggestions for additions and improvements. The unfamiliarity of the "machine accent" does pose a problem for first-time users, but we have found that only a short period of time is required before the users adapt their perception to the idiosyncracies of the synthesized speech. Our major applications thus far have been in three areas:

(1) Fortran Interpreter

We have adapted a locally designed Fortran interpreter to provide an interactive computer programming capability with voice output. Statements, data, and error diagnostic messages are all accessible through the synthesizer. The interpreter is used in teaching Fortran to blind students.

(2) Text Editor

An audio-output text-editing system has been developed which allows a blind user to prepare various kinds of texts: letter, reports, study notes, etc. The user can have the computer read back to him any portion of a stored text which he specifies. He can also delete, move, and insert material. When he wishes, he can dispose a copy of a text to the line printer to obtain an ink-print copy or record the audio version of a text onto a tape.

(3) Computer-assisted Instruction (CAI)

Several experimental lessons in the elements of computer science and linguistics have been written in the PLANIT CAI language, with all the computer-generated messages heard over the synthesizer. Students attending the Michigan School for the Blind, Lansing, have tried these lessons out.

IMPLICATIONS FOR NEW COMPUTER LANGUAGES

Probably the most fascinating aspect of our experience in using this voice output system is the impetus which it has given us to constructing a special computer language for handling non-visually presented material.

In designing a computer communications system for the blind, it is not enough merely to give the printed output characters a voice. Rather, we must examine what elements of computer output format are oriented specifically to the visual channel of perception and which are not. And we must turn our attention to ways of using the audio channel so as to take advantage of its capacity for carrying information.

It is not unreasonable to say that most of our present-day formatting conventions are oriented to the visual appearance of the printed page. Columns of numbers present themselves to the eye as units of form. Skipped lines and indentations set units of text apart from one another. Given the potentials of spoken output, what kind of auditory formats can we employ to create similar effects for arranging data? Pauses, intonational pitch contours, stress patterns, vowel length and vowel quality are devices which human language uses to superimpose formal structures on sequences of segmental phonemes. Just as a printer uses different type faces to set off parts of a text without disturbing the actual spelling of words, a voice output format system for a computer can range over several "dialect" styles to indicate variations in the status of output data. Cantillation styles or melodies can be designed to facilitate recognition of text units.

The orthographic system used in everyday English writing is an abstraction of certain features of speech, and the amount of the speech signal which goes unrepresented by our spelling is quite substantial. By and large, our spelling does not take account of interspeaker differences in pronounciation, speech rate, emotional overtones, and various other prosodic features. In reconstituting the acoustic signal via a voice synthesizer, we have considerable choice in the parameters which relate to this indeterminate portion.

It is important, too, to recognize that the very terms *voice synthesizer* and *speech synthesizer*, when applied to a specific device capable of delivering a finite repetoire of sounds in unlimited permutations, are really too restrictive. Just as an automobile is far more than a "horseless carriage" and radio is far more that "wireless telegraphy", the voice or speech synthesizer need not be seen only as a device for simulating human speech. At this point, one can only speculate on the kinds of new "languages" which may grow from man's encounter with a machine that talks, but it is clear that the speech synthesizer can be used as an instrument for exploring alternate modes of encoding information: new symbol systems which share some of the characteristics of speech, but which are specifically suited to the computer's powers and limitations, just as natural human languages reflect in their structure the inherent mental and physical powers and limitations of human beings.

One important benefit of this exploration of alternative auditory symbol systems lies in the development of sensory aids for the blind. At the time of Louis Braille, the blind were taught to read by feeling the shapes of large letters embossed on paper, if taught at all. Braille's brilliant achievement consisted in taking man's natural sense of tactile discrimination and developing out of it a symbol system which bears little physical resemblance to the visually-oriented writing system. In so doing, he greatly enhanced the ability of the blind to communicate. In the same way, we hope that the development of voice output systems for computers will lead to fuller utilization of the blind persons cognitive abilities.

REFERENCES

1. Chapman, W. D., "Prospectives in voice response from computers," *Proc. Internatl., Conf. Commun.*, 1970.
2. Cleave, John P., "Braille Transcription", *Mechanical Translation*, II: 3, December, 1955.
3. Cooper, F. S., J. H. Gaitenby, I. G. Mattingly and N. Umeda, "Reading aids for the blind: a special case of machine-to-man

communication", *IEEE Trans. Audio and Electroacoustics*, Vol. AU-17, pp. 266-270, December 1969.

4. Flanagan, J. L., *Speech Analysis, Synthesis, and Perception*, New York Academic Press, 1965.
5. Flanagan, J. L., C. H. Coker, L. R. Rabiner, R. W. Schafer and N. Umeda, "Synthetic Voices for Computers". *IEEE Spectrum*, Vol. 7, No. 10, pp. 22-45, October, 1970.
6. Gammill, Robert C., *Braille Translation by Computer*, Report No. 9211-1, Department of Mechanical Engineering, Massachusetts Institute of Technology, Contract No. SAV-1-11-62, October, 1963.
7. Holmes, J. N., I. G. Mattingly and J. N. Shearme, "Speech synthesis by rule", *Language and Speech*, Vol. 7, pt. 3, pp. 127-143, July-Sept., 1964.
8. Kelley, J. L., Jr., and L. J. Gerstman, "An artificial talker driven from a phonetic input," *J. Acoust. Soc. Am.*, Vol. 33, p. 835 (A) 1961.
9. Lee, F. F., "Reading Machine: from text to speech", *IEEE Trans. Audio and Electroacoustics*, Vol. AU-17, pp. 275-282, December, 1969.
10. Nye, P. W., J. D. Hankins, T. Rand, I. G. Mattingly and F. S. Cooper, "A Plan for the Field Evaluation of an Automated Reading System for the Blind," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-21, pp. 265-268, June, 1973.
11. Rahimi, M. A. and J. Bryson Eulenberg, "A Computer Terminal with Synthetic Speech Output," *Behavioral Research Methods and Instrumentation.*
12. Schack, Ann S. and R. T. Mertz, assisted by Fred Brooks, *Braille Translation System for the IBM 704*, Preliminary write-up, 1961, International Business Machines Corporation.

A computer-based system of speech-training aids for the deaf—A progress report*

by R. S. NICKERSON, D. N. KALIKOW, and K. N. STEVENS

Bolt Beranek and Newman, Inc.
Cambridge, Massachusetts

This paper is a progress report on an effort to develop a computer-based system of speech-training aids for the deaf. The project was begun with the assumption that an attempt to *design* such a system would probably fail, and that a more promising approach would be to attempt to evolve one through use. Accordingly, a system incorporating some of the capabilities that it was thought would be useful for speech training was developed, and installed at the Clarke School for the Deaf where it is now being used on an experimental basis in a remedial speech-training program. The expectation was that the capabilities of the system would be modified and extended as attempts to use it provided insights concerning what features it should have. To ensure that such insights do in fact guide the system's evolution, developers and users are engaged in a continuing dialogue concerning the desirability and feasibility of specific modifications and extensions, both in the training procedures that are used in conjunction with the system and in the characteristics of the system itself.

The general considerations that governed the initial development of the system were the following. Deaf students receive only minimal acoustic information from the speech of others and from their own vocalizations. The speech skills they acquire are based on cues they receive from their residual hearing and from visual observations of the gestures of others. Often these skills are inadequate and incorrect, and the students thus need special training in order to help them to produce intelligible speech. As a part of this training, it is customary for a teacher to produce speech-like patterns or to describe the patterns to the child and for the student to try to imitate these patterns. The student is encouraged by the teacher if he produces the correct speech gesture. Three problems arise in this kind of training situation: (1) the relevant attributes of the speech sample produced by the teacher often cannot be seen, felt, nor heard by the student; (2) the student must rely on the teacher to indicate whether or not his production is acceptable; and (3) the teacher must make a subjective judgment as to the adequacy of the student's production. All three of these problems provide motivation for developing a set of displays for use in a speech-training situation.

The idea of using visual displays of speech parameters to aid in speech training of the deaf is, of course, very old. In recent years, numerous instruments have been developed to produce a variety of different visual patterns.[1,2] Our system incorporates within a single unit some of the kinds of displays described previously by others (although usually in modified form), as well as some new displays.

The system is built around a small digital computer, the Digital Equipment Corporation PDP-8E. Speech information is obtained from a miniature accelerometer attached by thin double-stick tape either to the throat or the nose, and from a headmounted voice microphone. The accelerometer (BBN Model 501), which is approximately .3 inches in height and diameter, and weighs about 1.8 grams, is used to simplify the extraction of certain parameters that are relatively difficult to derive from a microphone output. When the accelerometer is attached to the throat it gives a waveform that has periodic peaks at the frequency of the glottal output during voiced sounds. The output is fed to a pitch extractor circuit that measures the time between positive-going zero crossings of the waveform and reports the pitch periods to the computer. When attached to the nose, the accelerometer provides a signal that is a measure of the amount of acoustic coupling to the nasal cavity through the velarpharyngeal port. In this case, the output, which is 10-15 dB higher when the velum is lowered—during nasalized sounds—than when it is raised, is fed to a component that rectifies and low-pass filters it, and sends the result on to the computer. The use of the accelerometer for the acquisition of pitch and nasality information is described more fully by Stevens, Kalikow, and Willemain.[3] The output of the voice microphone is fed into a filter bank that reports to the computer the energy in each of 19 frequency bands within the range 100-6560 Hz. Data from the pitch extractor or nasality circuit (only one of these components is operational at a given time in the current system) and the filter bank are sampled by the computer 100 times per second, and used to generate a variety of visual displays. Control inputs from the user are given to

* This is an abbreviated version of a paper that has been submitted for publication to *American Annals of the Deaf.*

the computer via a set of push-buttons and analog knobs. For further details concerning the system, see Nickerson and Stevens.[4,5]

Several different types of displays have been programmed. One provides the child with a game-like situation in which he can "shoot baskets" by performing certain vocal exercises. Another represents certain speech parameters in terms of changing features of a cartoon face. Still another provides the capability of displaying individual speech parameters (amplitude, voicing, fundamental frequency, nasality) either singly or in various combinations as time functions. The system continuously records, both digitally and on analog tape, the most recent two seconds of speech. Most displays, therefore, have freeze and replay capabilities. Some of the programs permit the teacher to produce a target pattern on the display which the child can then be asked to attempt to match. They also incorporate the capability of moving patterns about on the display so as to facilitate visual comparison of a representation of a student's utterance against a target that he may be attempting to match. In addition, they provide the means of showing, on request, the values of some of the parameters that are displayed.

Our initial experience with the system has been encouraging; however, it seems clear that how effective any speech-training aids will prove to be in practice will be bounded above by the specifics of the ways in which they are used. As technical developments make it feasible to do increasingly complex real time analyses of speech and to generate nearly anything one wants by way of displays, it becomes more and more apparent that pedagogical uncertainties impose the real limits on what one can expect to accomplish with speech-training aids, no matter how technologically sophisticated they may be.

A thought experiment demonstrates this point. Imagine a machine that could perform in real time any type of analysis of speech that one wished, and generate any display that one might specify. The fact is that we do not really know what analyses should be performed or what displays should be developed. Moreover, even if we knew the answer to these questions, it is not clear that enough is known about speech acquisition among the deaf to provide the basis for the training procedures that would take full advantage of such capabilities. What does seem clear to us is that the flexibility of a computer-based system provides opportunities for the type of exploration that is likely to be required to make progress on these problems.

Finally, the sort of close collaboration between researchers and teachers that we have attempted to maintain in this project is essential, we believe, if efforts to evolve effective training aids are to have a reasonable chance of success. This is not a new idea. Kopp[6] expressed the need for a greater interaction between teachers and researchers by suggesting that the field would benefit "if we could make more teachers researchers, and more researchers teachers." Other writers have also advocated such interaction,[7,8] but few serious attempts to collaborate seem to have been made. The strategy is a reasonable one, we feel, not only for the development of this particular system but for that of any complex system that is to involve a real time interaction between men and computers on problems for which approaches are not highly formalized and the solutions are not well understood. As David[9] has pointed out, the great versatility of the computer represents both an opportunity and a challenge. The opportunity is for creativity and innovation; the challenge is to be discriminating and practical. A close coupling between a system's developers and its users is perhaps the only way to assure a balance between innovativeness and practicality from which something both new and useful may emerge.

ACKNOWLEDGMENTS

Each of the following individuals has contributed significantly to the design, implementation, and use of the system described in this report: Robb Adams, Patricia Archambault, Arthur Boothroyd, Douglas Dodds, Ann Rollins, Robert Storm, and Thomas Willemain.

This project was sponsored by the U.S. Office of Education Media Services and Captioned Films Branch of the Bureau of Education for the Handicapped, under Contract No. OEC-0-71-4670 (615). We are pleased to acknowledge the encouragement and helpful suggestions of Lois Elliott, who served as contract monitor during the initial stages of the project.

REFERENCES

1. Levitt, H., "Speech processing aids for the deaf," *IEEE Transactions on Audio and Electroacoustics*, 1973, AU-21, pp. 269-273.
2. Pickett, J. M., "Recent research on speech-analyzing aids for the deaf," *IEEE Transactions on Audio and Electroacoustics*, 1968, AU-16, pp. 227-234.
3. Stevens, K. N., D. N. Kalikow and T. R. Willemain, "The Use of a Miniature Accelerometer for Detecting Glottal Waveforms and Nasality," Submitted to *The Journal of Speech and Hearing Research*, 1974, in press.
4. Nickerson, R. S. and K. N. Stevens, "An Experimental computer-based System of Speech Training Aids for the Deaf," in *Proceedings, Conference on Speech Communication and Processing*, Newton, Mass., April 1972, pp. 238-241.
5. Nickerson, R. S. and K. N. Stevens, "Teaching Speech to the Deaf: Can a Computer Help?" *IEEE Transactions on Audio and Electroacoustics*, 1973, AU-21, pp. 445-455.
6. Kopp, G. A., "The Application of Recent Findings in the Field of Speech Correction," *The Volta Review*, 1938, 40, pp. 638-640.
7. Borrild, K., "Experience with the Design and Use of Technical Aids for the Training of Deaf and Hard of Hearing Children," *American Annals of the Deaf*, 1968, 113, pp. 168-177.
8. Denes, P. D., "Speech Science and the Deaf," *The Volta Review*, 1968, 70, pp. 603-607.
9. David, E. E., Jr., "Speech in the Computer Age," *The Volta Review*, 1962, 64, pp. 394-397.

Computer-assisted instruction in mathematics and language arts for deaf students*

by PATRICK SUPPES and JOHN DEXTER FLETCHER

Stanford University
Stanford, California

INTRODUCTION

This paper summarizes a three-year project running from July 1, 1970 to June 30, 1973, which was concerned with research and development in computer-assisted instruction (CAI) for hearing-impaired or deaf students. CAI curriculums developed by the Institute for Mathematical Studies in the Social Sciences (IMSSS) at Stanford University were used by more than 1,000 deaf students during the 1970-71 school year and by more than 2,000 deaf students during the 1971-72 and 1972-73 school years.

THE STANFORD CAI SYSTEM

The central processor for the Institute's computer system is a Digital Equipment Corporation PDP-10. In addition to 256K of core memory, short-term storage of programs and student information was provided by sixteen 180,000,000-bit disk modules; long-term storage of student response data was provided by magnetic tape. Communication with remote student terminals in participating schools was provided by private telephone lines. High-speed data transmission (generally 2400 or 4800 baud) and time-division multiplexing were used to communicate with clusters of 16 or more student terminals. Of the more than 180 terminals connected to the Institute system in 1972-73, about 125 terminals could be used simultaneously with no appreciable detriment to the system's speed of response. Any curriculum could be run at any time on any student terminal.

Figure 1 shows a map of the United States on which superimposed lines indicate the network operating in 1972-73. As can be seen, high-speed data transmission to Austin, Texas and Washington, D.C. was used to distribute programs in the southwest and on the east coast. Also shown is a direct line to New Mexico, which supported a similar installation at Isleta Pueblo, an Indian reservation approximately 20 miles from Albuquerque, New Mexico.

The student terminals were Model 33 teletypewriters, which communicated with the central computer system at a rate of about 10 characters per second. In a typical school, one room containing 8 to 15 student terminals was assigned for CAI. Ordinarily one person was chosen by the school as the CAI terminal proctor; this same person was in charge of the equipment and the supervision of students in the terminal room.

When a student seated in front of a terminal presses the start key, the program responds by typing

```
HI
PLEASE TYPE YOUR NUMBER AND NAME.
```

Each student receives a number, which he inputs together with his first name. He uses the same number for all courses and types a one-letter identifier as a prefix to indicate which course he is requesting.

SUMMARY OF CAI CURRICULUMS

All CAI curriculums developed by the Institute were available to students in the participating schools for the deaf. The curriculums most relevant and most widely used were mathematics strands, arithmetic word problem solving, and a special language arts course developed solely for deaf students. In addition, a basic English course (available from Computer Curriculum Corporation), an algebra course, a computer programming course in AID, a computer programming course in BASIC, and a deductive logic and algebra course were used on various occasions by a number of students. A quantitative summary of usage for 1971-72 is shown in Table I.

We give here a brief description of the elementary mathematics curriculum and the language arts curriculum.

Elementary mathematics strands

The objectives of the curriculum were (a) to provide supplementary individualized instruction in elementary mathematics at a level of difficulty appropriate to each student's level of achievement, (b) to allow acceleration in any concept area in which a student demonstrates proficiency and repeated drill in areas of deficiency, and (c) to provide a daily profile report of each student's progress through the curriculum.

* This research was supported by Office of Education Grant OEG-O-70-4797(607), OE Project No. 14-2280.

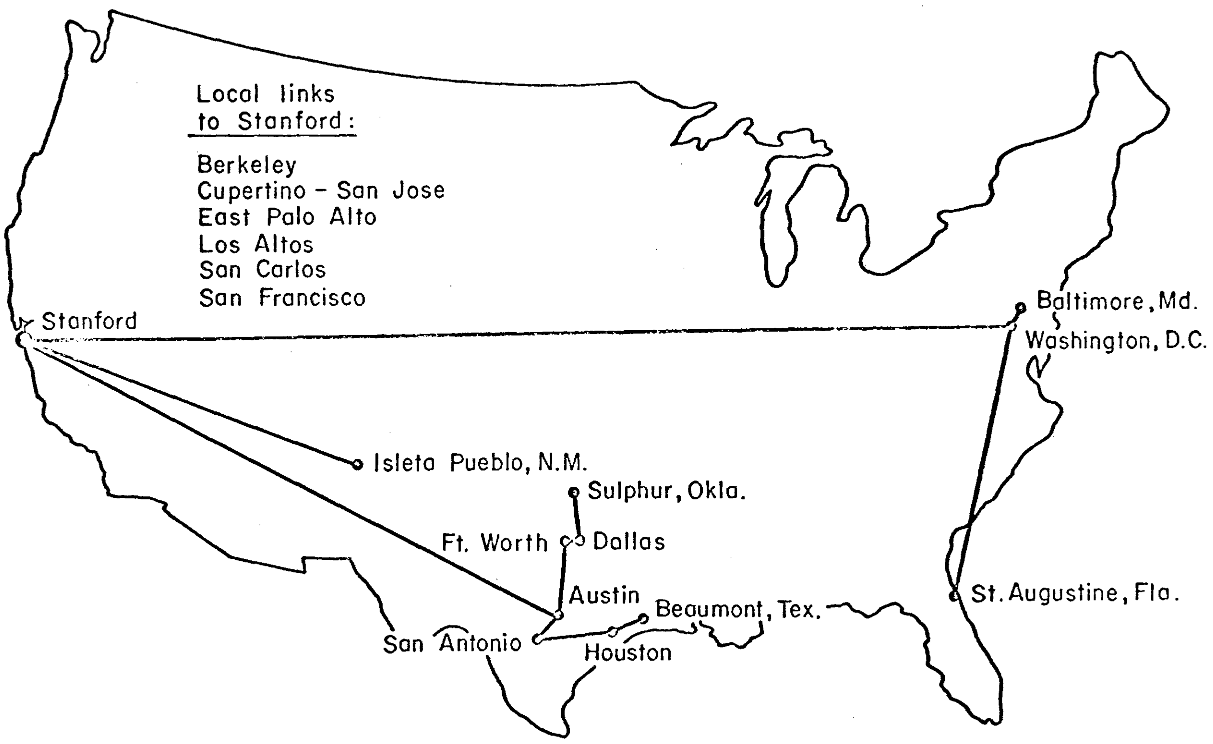


Figure 1—IMSSS national network, 1972-73

A strand is a series of problems of the same operational type (e.g., number concepts, addition, subtraction, fractions) arranged sequentially in equivalence classes according to their relative difficulty. The 14 strands in the program and the grade levels spanned by each strand cover the core elementary-school mathematics curriculum.

A student in the strands program works on fewer than 14 strands; the actual number depends on his grade level and performance. The strands approach provides a high degree of individualization because each student's lesson is prepared for him daily by the computer, the lessons are presented as mixed drills at a level of difficulty in each strand determined by the student's prior performance, and the student moves up each strand at his own pace.

Details of the curriculum are given in Suppes,[1] Suppes, Goldberg, Kanz, Searle, and Stauffer,[2] and in Searle, Lorton, and Suppes.[3]

Language arts

After carefully considering the language difficulties of hearing-impaired students, we designed the language arts curriculum to stress the structure of English, with particular emphasis on the roles of syntax and inflection and on the meaning of function words. An inductive rather than a deductive strategy was used. The course does not explicitly state 'rules' of English usage, rather it presents items illustrating aspects of standard English usage singly and in combination. Incidental learning of basic sentence patterns is enhanced by presenting curriculum items in complete sentences. Fewer than one-tenth of the exercises present the student with single words or isolated phrases. Incidental learning is also enhanced by requiring many constructed rather than multiple-choice responses.

There are four general course objectives. Students are to:

(1) Recognize specified grammatical categories;
(2) Recognize and supply various forms of given grammatical structures;
(3) Select appropriate grammatical units to complete a specified structure; and
(4) Perform specified transformations on grammatical structures.

The curriculum is divided into 218 lessons of 20-30 exer-

TABLE I—Institute CAI Curriculums Used by Participating Schools for the Deaf, 1971-72

Curriculum	Number of students
Elementary Mathematics (Strands)	2146
Arithmetic Word Problem Solving	107
Language Arts	1071
Algebra	83
Basic English	165
Computer Programming in AID	93
Computer Programming in BASIC	124
Logic and Algebra	216
Total Students 2279	

cises. Separate topics are presented in separate lessons and often there is a sequence of lessons on a single topic. The lessons are ordered to provide a cumulative basis of concepts building upon one another. Several lessons review topics presented in preceding lessons.

The course was described in detail by Fletcher and Beard,[4] Fletcher, Jamison, Searle, and Smith,[5] and Fletcher and Stauffer.[6]

It should be emphasized that the network was primarily developed to bring elementary mathematics and language arts to deaf students. Use of the other courses was on an optional and relatively infrequent basis in relation to the total number of students, but the network was flexible enough to provide additional work for students who wanted it, ranging from a secondary-school course in English to computer programming.

EVALUATION OF ACHIEVEMENT

During the course of the three years of the project a number of detailed studies were undertaken to measure the achievement of students using the CAI courses. We summarize here the two main studies dealing with achievement in the elementary mathematics strands curriculum and in the language arts curriculum.

Mathematics strands experiment

The purpose of the experiment was to measure the effect of varying numbers of mathematics strands sessions on arithmetic computation grade placement (GP) measured by the strands curriculum and by an on-line, computer-administered version of the Stanford Achievement Test (SAT) Arithmetic Computation subscale. This on-line version of the SAT was called the Modified SAT or MSAT. Construction and administration of the MSAT was detailed by Suppes, Fletcher, Zanotti, Lorton, and Searle.[7] Each student was allowed to take only a specified number of mathematics sessions at the terminal. All other sign-ons were spent working language arts lessons.

Three hundred eighty-five students from among those who were taking both CAI mathematics strands and CAI language arts, whose average GP on strands was between 2.4 and 5.9, and who had taken at least 15 mathematics strands sessions, began the experiment. The students selected were assigned at random to five experimental groups that differed in the maximum number of mathematics strands sessions they permitted during the experimental period of approximately 70 school days. Treatment groups 1, 2, 3, 4, and 5 were assigned 10, 30, 70, 100, and 130 sessions, respectively.

Session limits were imposed on a calendar basis so that students with low numbers of sessions received them distributed throughout the experimental period. A participating student had no control over whether he received a mathematics strands or language arts lesson. Whether he signed on for mathematics strands or language arts a student was given a mathematics strands lesson if he was eligible for one. Otherwise, he received a language arts lesson.

Five models were tested to study the relationship between the two independent variables of pretreatment scores and the number of mathematics strands sessions on the one hand and the dependent variable of posttreatment scores on the other. We tested a linear regression model in the two dependent variables, a linear regression model with an interaction term between the two independent variables, a multiplicative Cobb-Douglas model of econometric type, a log-quadratic model in the two independent variables, and an exponential model in the two independent variables. Detailed results are not summarized here.

Parameters for the five models were generated twice, once using mathematics strands average GP as pretreatment and posttreatment achievement measures and once using MSAT GP scores. The linear model with interaction accounted for more of the variance in the dependent variable (posttreatment average GP) than did any of the other models, but despite the inclusion of a term for the interaction of number of sessions with pretreatment GP, it represented only a slight improvement over the simple linear model. Assuming $N_i=120$ or slightly less than one session per day for a school year and taking $a_2=.0123$ from the linear model, we can project $T_{i2}-T_{i1}=1.48$. That is to say, if a student from this population takes about one strands session per day for an entire school year, we can expect his strands average GP to increase by about a year and a half. Data presented later show that strands average GP underestimated both GP measured by paper-and-pencil administrations of the SAT and GP measured by the MSAT. This improvement of 1.48 can be compared with an expected GP increase over a school year of .3 to .4 in the SAT computation subtest for hearing-impaired students receiving ordinary instruction.[8]

Among the models and parameters using MSAT GP as pretreatment and posttreatment measures, the multiplicative model from econometrics that assumed weighted interaction of number of sessions with pretreatment GP accounted for more of the variance in the posttreatment measure than did any other model, but, as with strands average GP, it represented only a slight improvement over Model I, the simple linear model. Again, assuming $N_i=120$ and taking $a_2=.0084$ from the linear model, we can project $T_{i2}-T_{i1}=1.01$. That is to say, if a student from this population takes about one strands session per day for a school year of 120 net days, we can expect his MSAT GP to increase by about one year. Roughly, we can expect an increase of .1 in MSAT GP for every 12 sessions taken.

Suppes, Fletcher, Zanotti, Lorton, and Searle[7] concluded that the mathematics strands CAI curriculum can lead to substantial increases in mathematics computation GP when used by hearing-impaired students. The increases are sufficient to bring the students to GP gains expected of normal-hearing students. Moreover, these gains can be achieved by students working intensely for only a few minutes a day in a supplementary drill-and-practice program. The time spent at a computer terminal by each student ranged from 6 to 10 minutes for each session.

In addition, Suppes, Fletcher, Zanotti, Lorton, and Searle[7] concluded that a simple linear model of student achievement gives a good account of the posttreatment distribution of GP measured either by the MSAT or by the strands GP. The investigation of other models, including models with interaction terms, did not lead to any substantial improvement in accounting for posttreatment GP variance. The results of the analysis, including the application of the linear model, indicate that greater numbers of CAI sessions are beneficial for all students, across all levels of pretreatment achievement.

Language arts experiment

This experiment was analogous to the mathematics strands experiment described by Suppes, Fletcher, Zanotti, Lorton, and Searle.[7] Each student was allowed to take only a specified number of language arts sessions. All other sign-ons were spent working mathematics strands sessions.

Two hundred thirty students from among those who were taking both CAI mathematics strands and CAI language arts in 1972-73 were selected for the experiment, and were assigned at random to one of five experimental groups that differed in the maximum number of 10-minute language arts sessions they permitted. Students assigned to groups 1, 2, 3, 4, and 5 were permitted 20, 45, 70, 95, and 120 sessions, respectively. The subjects were selected from students in the California School for the Deaf, Berkeley, California; the Oklahoma School for the Deaf, Sulphur, Oklahoma; and the Texas School for the Deaf, Austin, Texas. Random assignment of these subjects to the five treatment groups was stratified so that roughly the same number of students from each school could be assigned to each of the treatment groups. When the experiment began, 45 students were assigned to group 1, 46 were assigned to group 2, 46 were assigned to group 3, 47 were assigned to group 4, and 46 were assigned to group 5. One-way, fixed-effects analysis of variance and five models of student progress were used to investigate student performance at the end of the 80-school-day experiment period. The five models of student progress investigated were the same as those used in the mathematics strands experiment.

The assistance of teachers and proctors was sought to help students achieve the number of language arts sessions they were assigned. Teachers were urged not to give compensatory off-line work to those students assigned to low numbers of on-line sessions, and, in general, not to alter the classroom work of any student because of his participation in the experiment.

Fletcher and Beard[4] reported that complete data were obtained for 197 subjects. However, 46 of these subjects had received 100 or more sessions in 1971-72 and these subjects were removed from the experiment prior to any data analyses which were then performed on the 151 remaining subjects. In the analysis of variance there were 33, 27, 26, 33, and 32 subjects in treatment groups 1, 2, 3, 4, and 5, respectively. Students in groups 1, 2, 3, 4, and 5 received an average of 22, 46, 69, 88, and 106 sessions, respectively. These averages were lower than expected for groups 3, 4, and 5, but the treatment groups appeared sufficiently distinct to proceed with analysis of variance. The F-ratio for this analysis was not statistically significant, indicating that the range of sessions considered did not have a significant effect on posttest scores. The paper-and-pencil language arts test developed by the project appeared to be reliable and fairly valid. The correlation between pretest and posttest scores on the test was .910 with an F-ratio for significance of regression beyond $p<.01$, and the correlation between posttest scores and number of lessons completed was .645 with an F-ratio for significance of regression beyond $p<.01$.

Models I, II, III, IV, and V accounted for 83 percent, 83 percent, 66 percent, 83 percent, and 33 percent, respectively, of posttest score variance. The only model to which a term that included a measure of sessions taken contributed significantly was Model V. In all other models the only significant independent variable was the pretest score. An additional model, Model VI, was investigated. This model was of the form

$$E(T_2)=a_0+a_1T_1+a_2N+a_3L,$$

where T_2 refers to posttest score,

T_1 refers to pretest score,
N refers to number of sessions taken,
L refers to number of lessons completed,

and a_0, a_1, a_2, and a_3 are parameters of the model. Model VI accounted for 85 percent of the variance in posttest scores. Both sessions and lessons, in addition to pretest scores, contributed significantly ($p<.01$) to the model. However, the regression coefficient in Model VI for number of sessions taken was negative, indicating an inverse relationship between number of sessions taken and posttest scores when number of lessons completed was taken into account.

Fletcher and Beard[4] concluded that the course is of significant value to students whose ratio of lessons completed to sessions taken is high but of much less value to students whose ratio of lessons completed to sessions taken is low. The relationship between sessions taken and posttest scores was concluded to be more complex than anticipated.

Language arts item analysis

Fletcher and Beard[4] reported several results from their item analysis of the language arts curriculum that are not widely noted in the literature on deafness.

First, the "directions" lessons were far easier than anticipated, given the general impression among deaf educators that deaf students experience difficulty in following directions. Some reasons for this result may be that the directions in these lessons and in the curriculum were easier to follow than those given in classroom instruction, that the directions given in the language arts CAI were more clearly communicated to students than the directions given in classroom instruction, and that deaf students have less difficulty following directions than generally supposed. More research is required to decide among these alternatives.

Second, although pronouns were generally far easier than anticipated, items on possessive pronouns were extremely difficult for the students. Specifically, possessive pronouns that differ in number (his boxes, their box) and/or gender (his sister, her husband) from the nouns they modify were seldom completed correctly.

Third, copulas joining subjects with predicate complements that differ in number from their subjects were very difficult for the students. Copulas for items such as the following:

> The house (is, are) blue and white.
> The girls (seem, seems) lonely.

were seldom completed correctly.

Fourth, the students had very little trouble with contractions with the exception of "I'm," which was far more difficult than anticipated.

CONCLUSIONS

We began this project with the conviction that we had a powerful instructional tool at our disposal. Our aims were to demonstrate that CAI could be used to advantage by deaf students, that it could support serious research in deaf education, and that a favorable argument could be made for the economics of CAI. Behind these aims was the general intent of initiating large-scale use of CAI in schools for the deaf. To some extent we successfully met each of these aims.

It seems reasonable to conclude that CAI can be used successfully by deaf students. We did not set out to apply CAI to all of deaf education; we attempted only what we could do well. The curriculums concentrated on the skill subjects of mathematics and language arts, and, within these subjects, we emphasized aspects that were most amenable to computer presentation. Under these constraints we achieved favorable results. Certainly, the gains in mathematics computation ability that were two to three times greater than those expected from classroom instruction and the precision with which GP increase could be predicted as a function of CAI sessions are notable.

We also concluded that CAI provides a substantial foundation for research on the problems and processes of deaf education. The range of research undertaken by this project barely represents the diversity of inquiry that can be supported by CAI. The unobtrusive and precise control over experimental conditions made possible by computer presentations, as well as the accuracy and speed of computer arithmetic and data retrieval, permits a wide spectrum of experimental possibilities that we have only begun to explore.

The major drawback of CAI, however, is its cost. Computers require a sizable commitment of funds, both for acquiring capital equipment and for maintaining operations. Fortunately, the steady increase in the quality of available CAI is matched by a steady decrease in its costs. In the mid-1960's, when CAI first became available, it cost about $40 per student contact hour. Currently, CAI offered by the IMSSS system costs $1.50-$2.50 per student contact hour, depending on communication expenses. For the immediate future we can expect continued decreases in the costs and continued increases in the quality of CAI.

The proof of this project is in its impact on deaf education. Specifically, the willingness of the participating schools to support CAI from their own funding sources is the ultimate test of the project's impact. To date 13, of the 15 schools that participated in this project have committed funds to continue their CAI activity in 1973-74. The two remaining schools have not decided what CAI implementation alternative to adopt. Two schools that received no CAI from this project will be added to those supporting CAI in one network that directly resulted from this project. We expect the growth of CAI in schools for the deaf to continue.

REFERENCES

1. Suppes, P., "Computer-assisted Instruction for Deaf Students," *American Annals of the Deaf*, 1971, 116, pp. 500-508.
2. Suppes, P., A. Goldberg, G. Kanz, B. Searle and C. Stauffer, *Teacher's Handbook for CAI Courses*, Tech. Rep. No. 178, Stanford, California, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1971.
3. Searle, B. W., P. Lorton, Jr. and P. Suppes, *Structural Variables Affecting CAI Performance on Arithmetic Word Problems of Disadvantaged and Deaf Students*, Tech. Rep. No. 213, Stanford, California, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.
4. Fletcher, J. D., and M. H. Beard, *Computer-assisted Instruction in Language Arts for Hearing-impaired Students*, Tech. Rep. No. 215, Stanford, California, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.
5. Fletcher, J. D., D. T. Jamison, B. W. Searle and R. L. Smith, *Computer-assisted Instruction for the Deaf at Stanford University*, USOE Annual Rept., Stanford, California, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.
6. Fletcher, J. D., and C. M. Stauffer, "Learning Language by Computer," *The Volta Review*, 1973, 75, pp. 302-311.
7. Suppes, P., J. D. Fletcher, M. Zanotti, P. V. Lorton and B. W. Searle, *Evaluation of Computer-assisted Instruction in Elementary Mathematics for Hearing-impaired Students*, Tech. Rep. No. 200, Stanford, California, Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.
8. Gentile, A., and S. Di Francesca, *Academic Achievement Test Performance of Hearing-impaired Students*, Series D, No. 1, Washington, D.C., Office of Demographic Studies, 1969.

Integrated voice/data compression and multiplexing using associative processing*

by LEON D. WALD

Honeywell Systems and Research Division
Minneapolis, Minnesota

INTRODUCTION

The Associative Communications Multiplexer (ACM) concept is based upon the Associative Processor (AP). For the purposes of the following discussion, an AP will be defined as an ensemble of processing Elements (PEs), each capable of storing information, comparing that information with an internal or externally applied comparand argument, and performing arithmetic operations between the various internal and external operands. The same operation—store, compare, or process—may be executed on the differing local data, simultaneously in all or any subset of processing elements. A single control unit, with internal program memory, supplies instructions in parallel to all PEs to control their operations.

Associative processing can be effectively applied wherever a set of common operations must be performed on many pairs of operands. Numerous instances of this requirement may be found in the area of voice and data communications. The Associative Communications Multiplexer has been designed to apply these techniques in three functional areas, which are described below.

Digitization and loading of multichannel input data

Since an associative processor can execute calculations simultaneously for all elements, data I/O often must also be parallel to retain the speed advantage. In the ACM, this capability is utilized to simultaneously sample and digitize all analog input channels. At the same time, digital channels are also sampled.

Simultaneous compression and concentration of multiple channels

The ACM also applies associative processing to compression and concentration of multichannel data. A number of parallel algorithms may be used to compute figures of merit for all channels which represent relative significance of the data at a particular sample time. The calculation tends to yield high figures of merit for active channels and much lower values for those, including digital channels, in the pause mode.

Parallel selection of most-significant data

The parallel search capability of the ACM is used to select the most significant data for transmission each cycle. Figures of merit for all channels are compared by priority group and those with highest priority and activity are transmitted, up to the channel capacity. Thus, dynamic asynchronous multiplexing is achieved wherein the compression and quality of analog channels are automatically adjusted to account for changes in overall system activity, while keeping the high speed channel loaded to capacity and without necessitating buffering or delay.

ORGANIZATION OF THE ACM

The ACM is comprised of three subsystems as indicated in Figure 1. The I/O subsystem performs the interface functions, sampling, digitizing, and buffering. This unit also reconstructs the analog signals after transmission, and resynchronizes digital channels. The Associative Processor (AP) executes the compression algorithm, and transfers the most significant samples to the output buffers, from which they are transmitted via the high speed digital data link. The control unit provides overall system synchronization and control and features stored programs, both at the basic operation (microinstruction) and algorithm (main program) levels, for maximum flexibility.

The architecture is a highly modular ensemble of nearly identical computing elements. Although several versions of the I/O circuitry are necessary to interface the various analog and digital sources, each Processing Element (PE) is identical to all others and is comprised of at least one byte of Associative Memory (AM), a few registers, an adder, and a number of Random Access Memory (RAM) bytes.

Two detailed organizations have been considered for the

* This work was supported in part by the Rome Air Development Center, Contract No. F30603-71-C-0350.

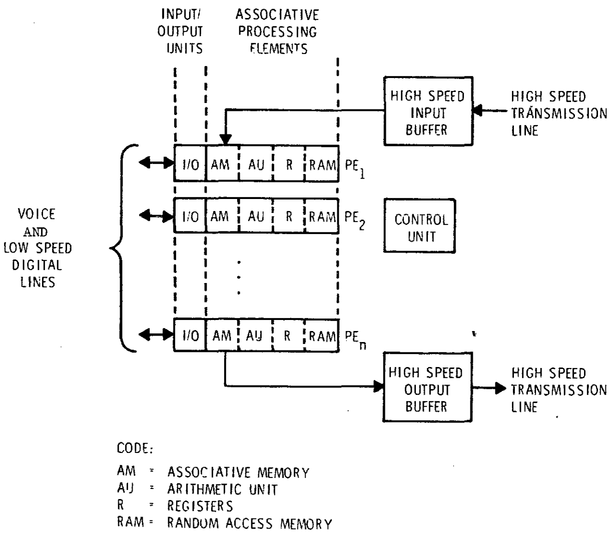


Figure 1—Associative communications multiplexer block diagram

PEs. The least complex combines a fully parallel AM with bit slice arithmetic and access to the RAM, and would be adequate only for the simplest compression techniques. The second PE architecture permits data transfer and arithmetic to be parallel by 8-bit bytes and is thus much faster.

The parallel PE organization is shown in Figure 2. The associative memory is not shown explicitly, since the adder/subtractor performs the associative comparison function required for this application. Essentially, the RAM is made to look associative. Compared to a conventional AM, the adder offers the advantage that inequality (such as greater than) searches can be accomplished as rapidly as equality searches. To maximize this advantage, the global data buss has been given direct access to the adder.

The RAM, which is addressed from the control unit using the RAM Address Register (RAAR), would typically store 32 8-bit bytes. The data from both registers, from the Input/Output Processor (IOP), and from the global data lines can be stored in memory by using the adder output bus. This implementation allows the locations within memory to be treated as registers, permitting operations that take a word from memory, add or subtract a word from a register or the input lines, and put the result back in the same memory location.

Local operands are placed in the two general registers, TRR1 and TRR2. The latter has a built-in shift capability. The tag register, 4 bits in length, permits storage of information such as connect status and search response history. The Element Activity (EA) flip-flop state may be modified under control of these tags.

A number of simple logic blocks (L1, L2, and L3) aid in control of the element. These function to indicate zero, limit, and overflow conditions, aid in shifting and rounding, and control the activity commands.

The ACM system, and particularly the PEs, when built, will make extensive use of medium and large scale integration. The ACM is organized so that channels may be added or deleted simply by modifying appropriately the numbers of each type of chip in the AP. Changes in programming or control unit hardware will not be required.

ACM OPERATION

Operation of the ACM may be clarified through consideration of the six basic steps in sequence as they are performed each cycle. First, in the *Measurement* step, input signals are sampled. Next, these samples are converted to digital form and stored in buffers in the *Conversion* step. During the third step, *Input*, the samples are transferred in parallel to the AP. Redundancy and pauses are removed in the fourth step, *Compression*. *Output*, the fifth step, involves selection and readout of the most significant data. Finally, in the *Reconstruction* step, analog waveforms are reproduced from the transmitted data and digital data streams are retransmitted into the low speed lines at appropriate data rates.

Since the Compression and Output functions are so basic to the ACM concept, these will be discussed in greater detail. Operation of a typical compression algorithm is illustrated in Figure 3, which shows several digitized samples for a single analog channel. This Zero Order Predictor (ZOP) algorithm begins with a transmitted point. A horizontal prediction line through this point and a horizontal corridor of width 2ε, symmetric to the point, are constructed. The values chosen for ε for each of the channels will determine relative speech quality levels. If a common value is used for all channels, then choice of this parameter is arbitrary.

As each succeeding point is received by the AP, the corridor boundaries are adjusted (reducing the corridor width) so that the upper and lower limits are respectively no more than ε above or below the point. The sequence is continued, even if the corridor width becomes negative, until

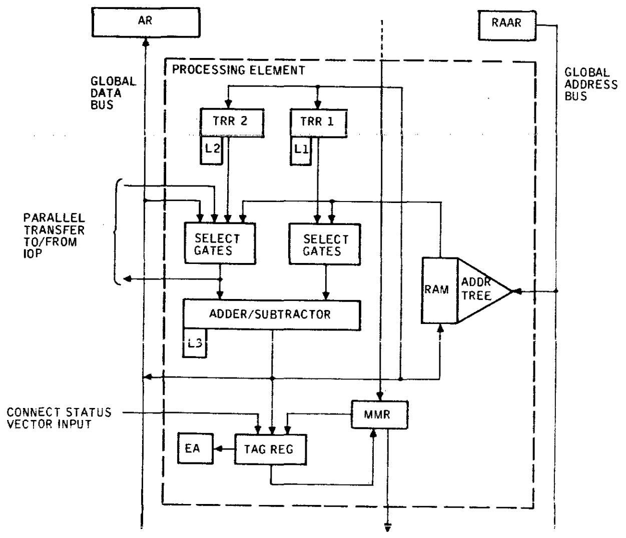


Figure 2—Byte-organized processing element

a transmission for that channel occurs. The process is duplicated for all other channels in the system.

The prediction lines essentially represent the output of the digital-to-analog conversion process at the remote ACM receiver assuming no transmission errors. The final reconstructed waveform (the approximation to the original input) is thus a filtered and smoothed version of the solid line shown in Figure 3.

It should be emphasized that nothing in the figure or the compression algorithm should be taken as a sufficient condition for transmission. Transmission may not occur even for negative corridor width or points outside the corridor, but depends entirely on activity relative to that of the other channels. For example, with reference to the figure, the following conditions where transmission occurred are illustrated:

- Point a—The sample is within a positive corridor.
- Point b—The sample is outside a positive corridor.
- Point d—The sample is outside a negative corridor.

At point c transmission did not occur, in spite of the fact that the corridor is negative and the point is relatively far from the prediction line.

This compression corridor, which decreases monotonically with time, is a measure of channel activity and may be used as an inverse figure of merit. In the ACM, the output or data selection algorithm compares all such widths by priority class during each cycle. In this way, data are transmitted from as many active channels in the top priority category down to a prescribed significance level as high speed output channel capacity will allow. The process is repeated successively for the lower priority classes as long as channel capacity remains. The result is a dynamic asynchronous analog/digital multiplexing technique which conserves transmission capacity by sending only the most significant information (i.e., channels exhibiting the greatest data change since the previous transmission).

A description of typical algorithm execution within the AP will serve to illustrate some of the ways in which associative and parallel processing can be used in compression

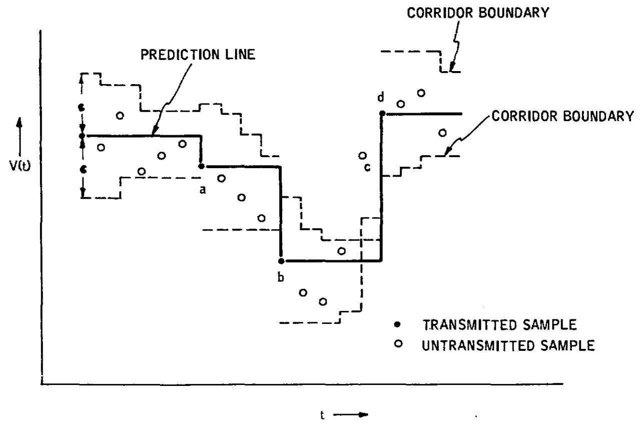


Figure 3—Zero order predictor compression algorithm

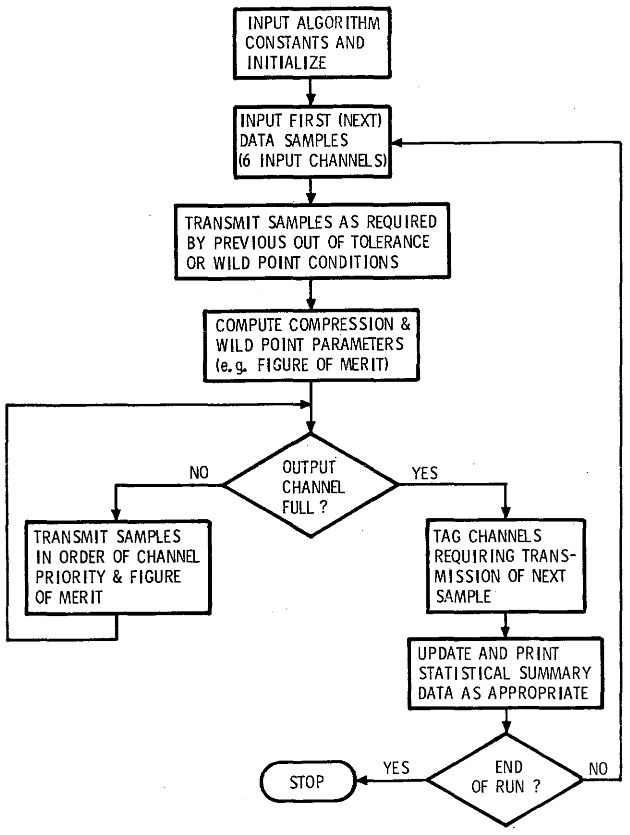

INPUT ALGORITHM CONSTANTS AND INITIALIZE
INPUT FIRST (NEXT) DATA SAMPLES (6 INPUT CHANNELS)
TRANSMIT SAMPLES AS REQUIRED BY PREVIOUS OUT OF TOLERANCE OR WILD POINT CONDITIONS
COMPUTE COMPRESSION & WILD POINT PARAMETERS (e.g. FIGURE OF MERIT)
OUTPUT CHANNEL FULL ?
NO
YES
TRANSMIT SAMPLES IN ORDER OF CHANNEL PRIORITY & FIGURE OF MERIT
TAG CHANNELS REQUIRING TRANSMISSION OF NEXT SAMPLE
UPDATE AND PRINT STATISTICAL SUMMARY DATA AS APPROPRIATE
END OF RUN ?
YES
NO
STOP

Figure 4—Associative communications multiplexer simulation

and multiplexing. For the AP, a processing cycle begins when the latest data samples are loaded into the RAMs in the corresponding PEs. To compress using ZOP, the sample in each PE is first subtracted from the current predicted value to determine whether it is above or below the prediction line. Then, the difference between the sample and the appropriate corridor boundary is compared with ε, and the boundary is modified if necessary. Completing the compression operation, the corridor width is formed as the difference of the boundaries.

Each of these operations is executed in parallel over as many PEs as required. Where a branch occurs in the algorithm, each PE determines the status of the appropriate condition and sets a tag accordingly. The control unit then issues a command setting the Element Activity (EA) flip-flops in the PEs conditioned by the tags, followed by a sequence of instructions corresponding to one of the branches. Thus, only the PEs required to execute the branch become active, and respond, in parallel, to the instructions. Later, the alternate branches are handled in an analogous manner.

Execution of the data selection routine is similar but is dominated by the use of the AP's associative search capabilities rather than by parallel arithmetic operations. All PEs are first asked to compare their channel priority codes to the code corresponding to top priority which is presented on the global data bus. PEs indicating a match are enabled, in

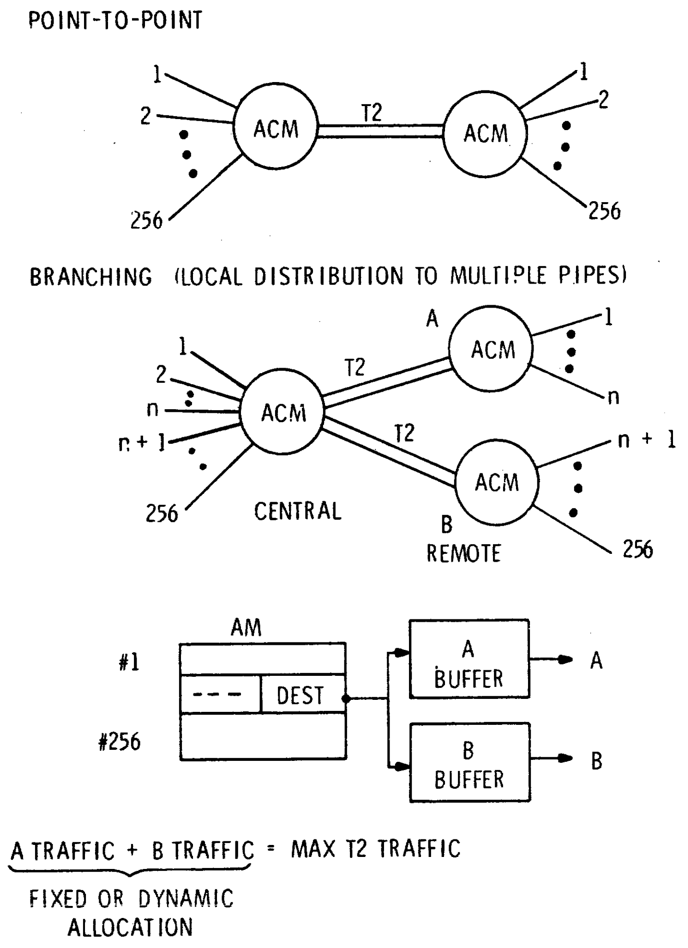

POINT-TO-POINT
ACM
T2
ACM
1
2
256
BRANCHING (LOCAL DISTRIBUTION TO MULTIPLE PIPES)
A
ACM
T2
n
n + 1
CENTRAL
B
REMOTE
AM
#1
#256
--- DEST
A BUFFER
B BUFFER
A TRAFFIC + B TRAFFIC = MAX T2 TRAFFIC
FIXED OR DYNAMIC ALLOCATION

Figure 5—Network configurations

sequence, for transfer of the data samples, via the global bus, to the high speed transmission buffer. The next lower priority code is then presented and the process is repeated. Within some priority classes, searches are also made for PEs characterized by corridor width below a specified threshold. This permits transmission ordered by channel activity levels but avoids the necessity for relatively slow minimum searches. When the transmission buffer is full, indicating that high speed channel capacity has been reached, data selection is terminated.

To measure the applicability of these and other compression and selection algorithms to actual voice data, a computer simulation has been written. This program, illustrated in block diagram in Figure 4, is written in FORTRAN and runs on an XDS 9300. It accepts 6 channels of digitized voice from magnetic tape as input, duplicates the ACM compression, noise suppression and output selection functions, and generates a 6-channel digital voice output tape. Since the input may be structured to be more active (with respect to pauses) than actual two-way telephone speech, ACM systems of up to 24 channels may be simulated. For example, if the six inputs include normal pauses between words and sentences but no time for the speakers to listen, then the system is processing twice the information normally characteristic of six channels, and is equivalent to a 12-channel system.

The simulator processes the voice data by examining, in sequence, the six sample values in each sampling period. Compression parameters such as corridor width are then computed for each channel. The current sample is also compared with recent history for that channel, and if outside prescribed limits, is designated "wild." Channels for which samples are not wild are ordered by corridor width. They are then successively tagged for transmission, beginning with the minimum width and proceeding until the assumed channel capacity has been reached. Those samples "transmitted" are transferred unchanged to the output tape. Samples for the remaining channels are calculated, as in the receiver ACM, in accordance with the reconstruction algorithm.

The ACM simulator has been used to measure the applicability of a number of compression and selection algorithms to speech. Among these are the Zero Order Predictor (ZOP) described earlier, the First Order Predictor (FOP) which uses sloping prediction lines and corridors, and combinations of the two algorithms. For these tests, actual speech segments for a variety of speakers and sentences were converted to

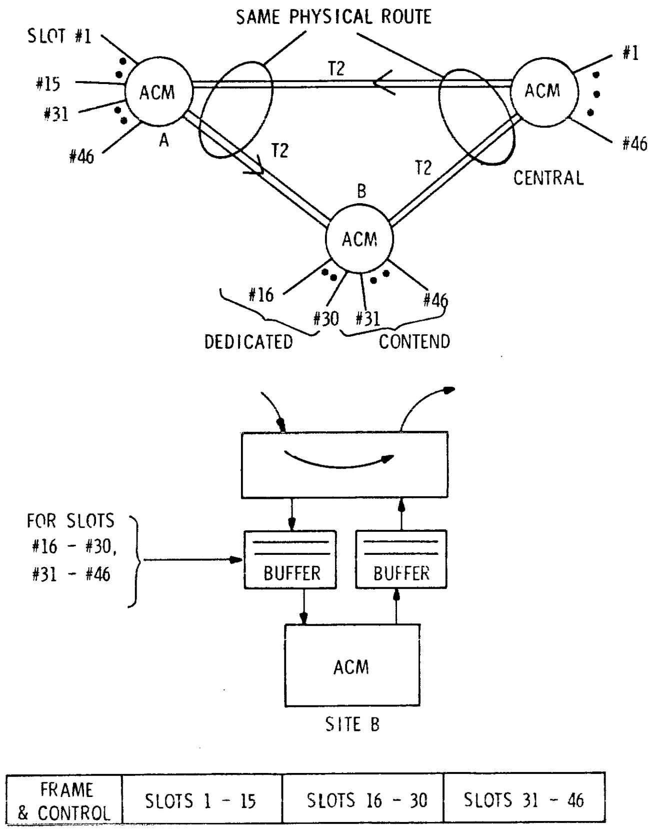


Figure 6—A multi-point network

8000 12-bit samples per second. Input channel timing was structured for several average activity levels and channel capacity was varied.

In general, test results were good, and support the validity of the ACM concept. Simple ZOP was found to yield the highest voice quality. Distortion and noise were found to be almost indistinguishable at a compression ratio of 4, acceptable at 8, and fairly severe at 12, the highest ratio used. Word and speaker recognition remained high even at maximum compression.

Although these speech processing experiments were by no means exhaustive or completely rigorous, they did lead to a number of interesting insights. For example, the superiority of a zero order algorithm over a first order one demonstrates that the theoretical ability of a piecewise linear approximation to more nearly fit the speech waveform does not result in higher quality. In fact, the studies showed that FOP tended to overshoot and hunt at waveform peaks and in regions of inactivity. This results in distortion, noise, and waste of channel capacity as extra points are transmitted to correct the errors. The latter produce additional distortion since other channels are hindered in their attempts to transmit at the most opportune times.

Detailed study of the way these algorithms operate upon actual voice has suggested some directions for improvement. A form of extrema detection algorithm, the Peak-Valley Interpolator (PVI) has been developed specifically for the ACM. Theoretically, PVI should result in higher compression by a factor of 2-3 and simultaneously in improved speech quality. Since it was not tested during this program, however, these conclusions remain to be validated.

ACM APPLICATIONS

The general applications area for the ACM can be defined as that where many channels of voice or a combination of voice and data must be transmitted efficiently over a costly high speed link. The ACM can fit into many communication system configurations.

The simplest ACM application is to the point-to-point system (Figure 5). Two remotely located ACM's are linked, and a one-to-one correspondence exists between channels of the one and those of the other. A given low speed channel on one end always corresponds to the same low speed channel on the other end. Switching between these channels can be added very simply (as an Associative Processor function) for a slightly more complex system.

A simple 2-branch network is also shown in Figure 5. On the basis of a destination address stored in the associative memory of the central ACM, each channel is directed through one of the output buffers and one high speed line to the appropriate remote ACM. Although a fixed channel allocation is shown, a dynamic one could be used as well.

Multipoint or loop applications are also possible, where several ACM's are interconnected as shown in Figure 6. Available time slots on the high speed line could be assigned dynamically on the basis of the relative activity at the various multiplexers. The technique shown assigns permanently a fraction of these slots to each remote ACM with remaining slots up for contention. Each ACM would attempt to use these contention time slots to transmit all its channels down to a prescribed activity level.

The ACM is capable of handling a variety of input types in almost any combination. These types include:

- Voice (analog or predigitized)
- Digital Data (real-time or non-real-time)
  - —Terminal-to-terminal
  - —Terminal-to-computer
  - —Computer-to-computer
  - —Telemetry
- Analog data
  - —Sensing and process control.

Included in this list are data for most of the commonly used information transfer applications, such as record message traffic, time-sharing systems, data gathering and computer load sharing.

Within the digital data category, both synchronous and asynchronous data can be handled. Low speed asynchronous data (to 600 bps) can be processed simply (but inefficiently) by treating the waveform as a compressible analog signal. This approach permits considerable flexibility for dialup voice/data terminals. Low to medium speed asynchronous data (to 2400 bps) can also be synchronized and buffered by character, automatically stripping off start and stop characters and deleting pauses. Information is thus accumulated at its normal (low) rate, transmitted in single character bursts at the ACM high speed output rate, and retransmitted by the remote ACM at the original data rate.

ACM treatment of synchronous data can be quite similar, in that the data can be buffered, concentrated, and transmitted at the high rate. However, since basic synchronous data rates are usually quite high (above 2400 bps) and the number of this type of channel will be quite low in a normal ACM application, it may not be worth the extra hardware to concentrate. Channels with data at rates below the maximum ACM input rate (64K bps) will of course be interleaved, achieving significant transmission efficiency.

CONCLUSIONS

The concept and design for the Associative Communications Multiplexer have been described, and applications to many areas of voice/data communications have been shown. The system is designed to allow economical transmission of voice and data in cases where cost of the transmission facility is dominant. Examples of such situations include satellite links, submarine cables, and certain long distance land lines.

A simulation of the ACM has been described which demonstrated many of the system's characteristics in processing speech. Listening tests, using actual voice inputs, confirmed that eightfold speech compression is possible and consistent with reasonable quality. These conclusions, and the effects of digital data, channel errors and many other factors are under more detailed investigation at the present time.

The ACM embodies a novel approach to the problems of compression and multiplexing since it combines digital and voice data in a single facility, and offers flexibility due to the system's modularity and programmability. Degradation due to overload is graceful and recovery is automatic and rapid. The system should ultimately be low cost, compact and reliable since its digital, modular architecture is ideally suited for large scale integration.

BIBLIOGRAPHY

1. Johnson, M. D., and D. C. Gunderson, "An Associative Data Acquisition System," *Proceedings of the 1970 International Telemetry Conference*, April 1970.
2. Wald, L. D., "An Associative Memory Using Large-Scale Integration," *NAECON '70 Record*, May 1970, pp. 277-281.
3. Wald, L. D., "An Associative Processor for Voice/Data Communications," *Proceedings of the 1972 Sagamore Computer Conference*, August 1972, pp. 135-144.
4. Hanlon, A. G., "Content Addressable and Associative Memory Systems—A Survey," *IEEE Transactions on Electronic Computers*, August 1966, pp. 509-521.
5. Stump, J. W., and L. W. Gardenhire, "Digital Voice Compression Study," *Final Report Contract DCA100-67-C-0018*, Radiation Incorporated Systems Division, October 1967.
6. Canover, M. F., "Investigation of Data Compression Techniques," *Final Report Contract NAS9-10876*, TRW Systems Group, 15 September 1971.
7. Ristenbatt, M. P. and D. R. Rothschild, "Asynchronous Time Multiplexing," *IEEE Transactions on Communications Technology*, June 1968, pp. 349-357.
8. VanBlerkom, R., G. R. Schwarz, and R. J. Ward, "An Adaptive Composite Data Compression Algorithm with Reduced Computation Requirements," *IEEE National Telemetering Conference Proc.*, 1968.

Speech as a man-computer communication channel

by REIN TURN

The Rand Corporation
Santa Monica, California

INTRODUCTION

Many computer applications require continuous interaction between men and computers. Typically, men communicate to computers data and programs, requests for processing and information retrieval, and other information required for the performance of computer-aided tasks. In turn, computers communicate to men the results of processing operations, the requested information, and any other messages they are programmed to produce.

The principal means for man-computer communication are manual, visual, and audio channels. The manual channel includes all mechanically operated input devices. The visual channel consists of printouts, displays and signals for visual sensing by man and electro-optical sensing by computers. The audio channels are the computer equipment and systems for recognizing spoken utterances, as well as the equipment for producing spoken output.

The choice of man-computer communication channels depends on numerous operational, human, and economic factors. Among these are the ease of use of the channel in the context of the man-computer tasks, the nature of the interaction language, the ability to maintain desired interaction rates, and the effects of the operational environment. The processing and storage requirements of the communication channel, and its cost-benefit advantages or disadvantages over competing channels, are important economic factors. An ideal channel is easy and natural to use, compatible with the total system, provides operational advantages and is cost-effective.

Most of the present man-computer communication channels are manual for man-to-computer communication and visual for computer-to-man communication. Their characteristics and design factors have been thoroughly analyzed and are widely available.[1]

The use of the speech channel is still in its infancy. However, the first generation speech synthesis equipment is becoming commercially available[2] and the current research in computer recognition of speech[3,4] is likely to make speech communication between man and computer technically and economically feasible in a few years. Limited capability isolated-word recognition systems are already being tested for simple control applications.[5] Several continuous speech understanding and recognition systems are being developed in the research laboratories.

Speech has the potential for becoming a versatile man-computer communication medium. This paper discusses its attractive features, problem areas, and application criteria for this purpose. A description of specific implementations of speech recognition systems, however, is beyond the scope of this paper.

SPEECH CHARACTERISTICS

It is a natural activity for a person to mentally encode his observations, ideas, and requests into a natural language—one that he uses in his daily communications with other persons—and express these in spoken form. Natural languages have evolved over long periods of time and, characteristically, permit great flexibility in expression and enormous variety in shades of meaning. That is, the mapping of mental images into natural language statements is a many-to-one process.

The expression of a given natural language statement in speech is another many-to-one transformation—the generated acoustic signals differ from speaker to speaker as functions of their voice tract physiology, age, sex, dialect, physical condition and emotional state.

The receiver of a spoken utterance must resolve the inherent uncertainties on the basis of context and his accumulated experience and knowledge. He uses his mental "model" of the speaker, the circumstances associated with the communication, and his "world model." Various non-verbal signals by the receiver also enter the understanding process. Sometimes the uncertainty cannot be resolved at all and further clarifying communications with the speaker are necessary.

The use of natural language utterances for speech communication with computers is beset with the difficulties outlined above. Since it is not practical to provide the computer with all the contextual information required to resolve the ambiguities, some restricted form of the language must be used. For example, the vocabulary may be limited to a few

* Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The Rand Corporation or the official opinion or policy of any of its governmental or private research sponsors.

hundred words that are used with unique meanings, and rigid syntactical rules may be imposed. Further constraints may be placed on the speakers (e.g., it may be required that isolated-word speech, rather than continuous speech, be used). Despite the loss in expressional power and flexibility that such restrictions entail, there are situations where speech is attractive for man-computer communication.

The following sections discuss the intrinsic characteristics and the associated attractive features and problem areas of speech input to computers. A part of this discussion is based on material which has previously appeared in literature.[6,8]

Message generation and encoding

The constant use of speech has made humans very skillful in communicating with others through this channel—speech can be produced effortlessly, spontaneously, at a high rate, and under almost all environmental conditions. Hence, the first characteristic of speech:

1. *Speech is man's natural and primary communication channel.*

The associated attractive features from the point of view of man-computer communication are:

- The use of speech is familiar and convenient when the interaction language is similar to the speaker's native tongue and is easy to pronounce.
- Speech is highly suitable and the preferred channel for spontaneously generated utterances.
- Speech is potentially the highest rate versatile communication channel for computer input.
- Using speech for man-computer communication may permit the "participation" of a computer in human discussions and teleconferences.

The speech input channel loses some of its attractiveness as the language departs more and more from natural language (e.g., when words are artificial and difficult to pronounce and when abbreviations, special characters, and punctuation marks are used). Some applications are not at all suitable for speech input, such as entering graphic data. Clearly it is more natural to trace out a curve on a graphic input tablet than to read the coordinates.

The potential speed advantage of spoken input is illustrated in Table 1. However, it must be borne in mind that a high *data* rate is not necessarily a high *information* rate.

The possibility of simultaneous communication with both men and machines has interesting implications. For example, a computer and its data base may become an active participant in a conference.

Interaction with other channels

The next speech characteristic pertains to its interaction with the other channels available for man-computer communication:

2. *The speech channel is independent of the visual channel or human voluntary motor activities (other than those required for speech production).*

The only muscles required for speech production are those that operate the vocal cavity, tongue, jaw and lips, and that control breathing. Other muscles and other bodily activities interfere only insofar as they affect breathing or require conflicting mental activities. Hence, an attractive feature is:

- Communication using speech can take place simultaneously with other visual or manual tasks, when the speaker is moving around, and in total darkness.

This is a very important feature of the speech channel. In numerous situations communication with the computer is not the only task. A standard example is piloting an aircraft while attempting to interact with the onboard computer.

Speech propagation

Speech propagates in the atmosphere in the form of pressure waves. These are reflected from and around objects. They can be easily changed into electrical form. The related speech characteristic is:

3. *Speech propagation is omnidirectional. No free line of sight is required.*

This leads to the following attractive feature:

- For speech input, the speaker can be in an arbitrary orientation relative to the microphone, at a considerable distance, or behind a barrier.

Microphones with various "fields-of-view" and sensitivities can be constructed and a computer input console would need not be user-centered, but could be "stretched out" to allow optimal placement of various input-output devices and displays. The user can walk around while entering informa-

TABLE I—Representative Data Rates for Man-computer Communication

Communication mode	Data rate (words/sec.)	Remarks
Oral reading[9]		
Random words	2.1-2.8	Selected from 5000 word dictionary
Random words	3.0-3.8	Selected from 2500 most familiar monosyllable words
Spontaneous speaking[9]	2.0-3.6	
Handwriting[8]	.3- .4	
Handprinting[8]	.2- .5	
Typing[10]	1.6-2.5	Skilled
Typing[10]	.2- .4	Inexperienced
Stenotype[11]	3.3-5	Chord typewriter
Touch-tone telephone[8]	1.2-1.5	
Thumb-wheel[12]	1.8 digits/sec.	Sequence of 10 digits
Rotary dialing[12]	1.5 digits/sec.	Sequence of 10 digits

tion through the speech input devices. The number and type of receivers may also vary:

- The speech target audience can vary freely from many (using loudspeakers) to a few (using earphones), in both cases low cost equipment can be used. This feature is important for security and privacy.

The easy conversion into electrical form leads to another attractive feature which has the potential of converting a conventional telephone instrument into a computer terminal:

- Speech communication with computers is compatible with existing voice communication networks and systems. This allows remote input from locations where no special computer-related equipment is available.

Among the problems associated with these characteristics is the interference of speech communications both by ambient acoustic noise and the electrical noise in the voice communication system. Another problem area is the transitory nature of speech—no hard copy is produced of speech input. A tape recording can be made but is inconvenient to use.

Speaker characteristics

The acoustic characteristics of speech signals depend on the structure of the speaker's vocal tract and its dynamics. Infections and other pathological conditions in the vocal tract also affect the speech quality. Articulation and timing are influenced by fatigue. Unusual emotional conditions can change the normal speech characteristics, such as the pitch and speaking rate. Hence the characteristic:

4. *Speech contains a great deal of information about the speaker.*

This characteristic leads to two attractive features and two problem areas in the application of speech for man-computer interaction:

- The use of speech allows checking the speaker's identity for access control purposes.
- The use of speech has the potential for monitoring the physical and emotional state of the user.

To implement speaker identification capability, carefully chosen speech samples can be analyzed and a set of parameters computed and stored. To authenticate a person's identity the person speaks a predetermined sentence which is also analyzed and the extracted parameters are compared with the stored ones. Considerable research is in progress on this topic.[13] An ability to monitor the operator's physical or emotional state[14] is important in man-computer tasks where the operator's actions, or inactions, may have drastic consequences, such as in air traffic control applications.

The problem areas associated with these features have to do with the complications in the design of speech recognition systems caused by speaker-to-speaker variability, and the variability in voice characteristics of a given speaker. It may be necessary to train the system to recognize each individual speaker's voice characteristics and to store such information in the system. Affected are the required amounts of storage and processing.

Environmental influences

Speech generation and propagation are both affected by the environmental conditions. Some of these, such as temperature, humidity, or insufficient working space, affect the speech generation only indirectly (e.g., by accelerating the onset of fatigue and emotional conditions); others have more direct effects. The associated speech characteristic is:

5. *Speech production is affected by mechanical forces on the speaker and composition of the atmosphere.*

Experiments have shown that both vibration and acceleration affect speech intelligibility.[15] Changes in the atmosphere, such as the presence of helium in submarine systems, also affect speech by changing the pitch and intelligibility.[16] However, weightlessness does not appear to have any effects.

Any ambient acoustic noise in the environment will interfere with the speech signal. This condition may be quite acute in systems containing equipment in operation (aircraft engines, teletype terminals) or other speakers. Among the techniques available for alleviating the noise interference problem are noise-cancelling microphones, special signal processing techniques, and the use of specially selected, high-intelligibility vocabularies.[17,19]

Speech in computer-to-man communication

Unlike the use of speech for computer input, automatic synthesis of spoken messages by computers is now practical. This is indicated by a recent survey of the state of the art in voice response systems[3] and by the number of firms actively engaged in marketing these systems.[2]

The attractive features of the use of speech for computer-to-man communication include the following:

- Speech is a natural way for humans to receive communications from others. It is compatible with the use of speech as a computer input channel.
- Several spoken messages can be received and comprehended simultaneously.
- Spoken messages can be received without interrupting the use of the manual or visual channels, in motion, or in total darkness.
- In receiving spoken output from the computer, the operator can be in an arbitrary orientation relative to the computer, some distance from the computer, or behind a barrier.
- Any number of listeners can receive the spoken message from the computer simultaneously.
- Speech reception by humans is not appreciably affected

by weightlessness, vibration, or mechanical forces on the listener.

There are also some problem areas. For example, the rate of receiving spoken messages is much slower than through the visual channel. The transient nature of speech requires that it be recorded on a tape if hard copy is required, but in this form it is not readily scannable by the human operator.

The ambient acoustic noise interferes with the reception and comprehension of spoken messages from the computer when they are broadcast or sent over a telephone instrument. However, the human auditory system is rather remarkable in its ability to select out and concentrate on a specific message and ignore others (this is the so-called "cocktail party" effect).

IMPLEMENTATION OF SPEECH INTERFACE

The design of an effective yet economical man-computer interface is a complex process that must take into account the nature of the tasks to be performed; the human roles, capabilities, and shortcomings in performing these tasks; the task performance environment; and the capabilities of the interface equipment. The task characteristics and the human roles determine whether a speech interface is suitable. The associated system design and performance requirements determine whether a speech interface will be technologically and economically feasible.

Design criteria

The principal roles of a human operator in a man-computer system are: decision maker, problem solver, controller, monitor, retriever or inquirer, and sensor or transducer. The most demanding of these is the human role as decision maker in real-time command-control systems where his performance is especially affected by the criticality of the consequences of the decisions, the diversity of the decisions to be made, and their dynamics.

The following considerations influence the design of a man-computer interface.

- Nature, time characteristics and variability of the tasks.
- Intensity level of the task performance and the operator's response requirement.
- Input and output loading of the operator.
- Operator's and system's physical state during task performance. Operator's physical safety and other stress conditions.
- Operator's level of isolation when performing the task.
- Environmental conditions.
- Training and skill level of the operator.

Based on these and the discussion of speech characteristics in the previous section the suitability of a speech interface for performing a given set of man-computer tasks can be evaluated.

The speech understanding and recognition systems used to implement a speech interface are characterized by a series of design features which reflect the acoustic and linguistic processing aspects of these systems. These design features have been discussed for continuous speech understanding systems in detail by Newell, et al.[8] Included are the following:

- Vocabulary size and syntactical structure.
- Number of speakers, their dialects and speaking habits.
- User training and system tuning; the degree of speaker-independence.
- Ambient noise environment and the transducer characteristics.
- Requirements for and availability of contextual "world model."
- Recognition error rate.
- Response time.

These establish the requirements for the system hardware—the special-purpose acoustic signal processing equipment and the general-purpose digital computer for pattern matching and linguistic processing. Tradeoffs can be performed between the various sets of the characteristics, especially between those involving the interface capabilities (vocabulary, syntax, speaker independence), performance (error rate, response time), and equipment (processing power, storage capacity, cost).

State of the art and potential applications

As mentioned previously, *isolated-word* speech recognition systems are already being offered on the market and are being tested. Typically, these systems can be trained to recognize utterances employing small vocabularies and highly restrictive syntax. The pause between words must be greater than .2 seconds.

The implementation of *continuous* speech recognition and understanding systems is much more difficult. One of the problems is the absence of word boundary indications in the acoustic signals and the dependence of the signal representing a word on the predecessor and successor words. This problem makes the use of linguistic and semantic information a necessity. The variability of individual speaking and articulation habits further complicates the recognition tests. Table II shows the recognition accuracy that has been achieved by various experimental and prototype speech recognition systems.[3]

There are a number of man-computer application areas, mainly in the "user's hands busy" category, where a speech interface for computer input-output could provide significant performance improvement. Among these are:

- Computer-aided fault diagnosis and isolation; computer-aided instruction; medical diagnosis; performance of scientific experiments.
- Data input in taking inventory, making observations, or tracking moving targets.

TABLE II—Speech Interface Performance Data

Researcher	Capability: Vocabulary	Capability: Speakers	Correct recognition percentage
Vicens (1969)	54 isolated	1	98-100
	54 isolated	10	79
	560 isolated	1	91
Yilmaz (1971)	16 isolated	10	99
Hill (1969)	16 isolated	12 unknown	78
Medress (1972)	100 isolated	5	94
Glenn (1971)	10 isolated digits	many	>99
Doddington (1973)	10 continuous digits	many	>99
Tappert & Dixon (1971)	250 continuous	several	75

- Monitoring computer-controlled processes.
- Controlling teleoperator systems and robots.

Other application areas are computer data base management, information retrieval, and computer-aided programming. The ultimate application is the perennial inventors' dream—the speech-operated typewriter for unconstrained language. It is not likely that such a device can be realized in the next decade, or even this century. Restricted versions, however, are likely to be implemented.

CONCLUDING REMARKS

The use of speech as a man-computer interface offers several attractive features over the conventional manual and visual channels. The most important among these are the independence of speech from the manual and visual channels which permits performing other tasks while communicating with the computer; the omnidirectional nature of the speech propagation, which permits the operator to use a computer while in motion or remote from the transducers; the ability to communicate simultaneously with men and computers; and the potential for using a telephone instrument as a complete computer terminal.

Despite the attractive characteristics of speech described in this paper, its use in a particular man-computer task makes sense only when its use is *natural* for performing the task and *compatible* with the environment. Hence the nature of the interaction involved must be thoroughly analyzed before committing to the use of a speech interface. However, together with other modes of man-computer communication, the speech-based interfaces can help an operator to concentrate on the tasks he is performing rather than on operating the interface equipment.

ACKNOWLEDGMENTS

The author would like to thank his colleagues at The Rand Corporation, Alan S. Hoffman, Robert C. Gammill, Gabriel F. Groner and Thomas F. Lippiatt for their helpful suggestions.

REFERENCES

1. Meadow, C. T., *Man-machine Communication*, John Wiley & Sons, New York, 1970.
2. Hornsby, T. G., Jr., "Voice Response Systems," *Modern Data*, November 1972, pp. 46-50.
3. Hill, D. R., "An Abbreviated Guide to Planning Speech Interaction with Machines; the state of the art," *Intern. J. of Man-Machine Studies*, Vol. 4, 1972, pp. 383-410.
4. Walker, D. E., "Automated Language Processing," in C. A. Cuadra (Ed.) *Annual review of information science and technology*, American Society for Information Science, Washington, D.C., 1973.
5. "Spoken Words Drive a Computer," *Business Week*, December 2, 1972.
6. Lea, W. A., "Establishing the Value of Voice Communication with Computers," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-16, No. 2, June 1968, pp. 184-197.
7. Hill, D. R., "Man-machine Interaction Using Speech," in *Advances in computers*, Vol. 11, Academic Press, New York, 1971, pp. 127-163.
8. Newell, A. et al., *Speech Understanding Systems*, North-Holland Publishing Co., Amsterdam, 1973.
9. Pierce, J. R., and J. E. Karlin, "Reading Rates and the Information Rate of the Human Channel," *Bell System Technical Journal*, Vol. 36, 1957, pp. 497-516.
10. Hershman, R. L., and W. A. Hillix, "Data Processing in Typing: Typing Rates as a Function of Kind of Material and Amount Exposed," *Human Factors*, October 1965, pp. 483-492.
11. Seibel, R., "Data Entry Through Chord, Parallel Devices," *Human Factors*, April 1964, pp. 189-192.
12. Deininger, R. L., "Rotary Dial and Thumbwheel Devices for Manual Entry of Sequential Data," *IEEE Transactions on Human Factors In Electronics*, September 1967, pp. 227-230.
13. Su, L., *On Speaker Identification*, Technical Report TR-EE 72-4, Purdue University, Lafayette, Indiana, January 1972.
14. Williams, C. E., and L. R. Simmering, "Emotions and Speech: Some Acoustical Correlates," *The Journal of Acoustical Society of America*, Vol. 52, No. 4, 1972, pp. 1238-1250.
15. Glenn, J. W., R. N. Gordon and G. Moschetti, *Voice Initiated Cockpit Control and Integration (VICCI) System Test for Environmental Factors*, Scope Electronics, Inc., Reston, Virginia, 20 April 1971.
16. Nixon, C. W., et al., "Study of Man During a 56-day Exposure to an Oxygen-Helium Atmosphere at 258 mm. Hg. Total Pressure: XVI. Communications," *Aerospace Medicine*, Vol. 40, No. 2,. February 1969, pp. 113-123.
17. Neely, R. B., and D. R. Reddy, "Speech Recognition in the Presence of Noise," *Working papers in speech recognition, I.*, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, April 21, 1972.
18. Drucker, H., "Speech Processing in High Ambient Noise Environment," *IEEE Transactions on Audio and Electroacoustics*, Vol. AU-16, June 1968, pp. 165-168.
19. Webster, J. C., and C. R. Allen, *Speech Intelligibility in Naval Aircraft Radios*, Technical Report NELC/TR 1830, Naval Electronics Laboratory Center, San Diego, Calif., 2 August 1972.

Verifiable secure operating system software*

by GERALD J. POPEK and CHARLES S. KLINE

University of California
Los Angeles, California

INTRODUCTION

While the desire for reliable security in multiuser computer systems has grown significantly, the computing community's knowledge of how to provide the requisite protection is still inadequate. Security is a "weak link" phenomenon; one link whose condition is unsatisfactory is the operating system software. It has often been pointed out that currently "no protection system implementation of any major multiuser computer system is known to have withstood serious attempts at circumvention by determined and skilled users."[1] The community is replete with apocryphal claims of secure systems that inevitably have failed.

Out of these difficulties and concerns has grown a good deal of activity. One part of the community has addressed the questions of how a system ought to be modularized, what its primitive elements ought to be, and what logical structure would be most useful. The useful concepts of capabilities and domains[2] have come from that activity. The Multics and Hydra systems are current examples of ongoing complex software systems in which serious attempts have been made to carefully design and structure the software with respect to protection considerations.[3,4]

Despite the value of these concepts and empirical laboratories, our knowledge concerning the *reliability* of protection systems is disturbingly inadequate. Currently, it is not possible to provide a meaningful guarantee that a system actually provides the controlled protection which the design claims. It is not possible to state with assurance that clever users will be unable to circumvent the controls, and thereby gain access to information, operations, or other resources which the design intended to prohibit.

What is required is a system which can do more than resist attacks by penetration teams. One would greatly prefer software which in a real sense has been proven correct with respect to certain precisely stated security predicates. Such a result would provide software protection of high quality, and help relieve fears that some feature or flaw had been overlooked.

At UCLA, a multiuser computer system is being constructed in which it is expected that verification of the security properties of the software will be successfully performed. That is, *meaningful and demonstrable operating system security is within our grasp.* While the system is not yet complete, work has progressed far enough that the viability and quality of the resulting system and proofs seem assured. In the following, we discuss concepts which have contributed to the system's design, as well as issues that arose during that design process. It is expected that the approaches described here have more general applicability.

Note that this work only concerns the protection enforcement mechanisms of the operating system software. Properly functioning, essentially error free hardware is assumed. The problem of authentication, that is, reliable identification of the user who presents himself to the system, also is not considered, except that a suitable environment is provided in which an authentication procedure could easily operate.

In addition, the flavor of protection toward which this work is initially directed is fairly simple. Mechanisms to support mutually suspicious subsystems, memoryless procedure calls, inference, or control of statistical access are not explicitly considered. Nevertheless, the insights provided by the goals of isolation and limited sharing apply to the more complex needs.

We now discuss a number of thoughts found relevant to the design of secure operating system software.

DEFINITIONS OF SECURITY

Before continuing to discuss the general facets of the UCLA design philosophy, it will help to explain a distinction we make between security and viability. We believe operating system security involves a set of essentially negative constraints. One desires to verify that certain actions cannot occur. For example, it will not be possible for the process associated with user x to have loaded with it a segment associated with user y. It will not be possible for users of class x to access files of class y.

One can coherently argue that such a point of view is incomplete; the null system satisfies negative constraints vacuously. What is required, the argument might continue, is inclusion of the idea of viable productivity—that the system actually supports useful activity. For example, if a

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under contract number DAHC-15-73-C-0368.

user could cause the process scheduler not to run any processes, would this not be a security flaw? This point is well taken. Certainly productivity is of importance since it embodies the primary reason for the existence of the system.

Nevertheless, we argue that a meaningful distinction can be made between the prevention of undesired actions and the need for productive activity. To have verified the negative constraints is a useful and nontrivial step, and those constraints certainly contribute to the overall viability of the operating system.

These negative constraints express the *security policy*. For each user, we can translate these negative constraints into a list of which security objects this user should and should not be able to access and with what access types. *Security objects* are the physical and logical parts of a computer system that need to be controlled, protected, or whose status needs to be guaranteed. An incomplete list of examples includes terminals, communication lines, processes, and files.

Security policy may be expressed in terms of *accessible sets*. For user A, *accessible*[t,A] is the set which defines what objects this user is to be allowed to access at time t. All other accesses are to be prohibited. Each entry in the set is an object, access type pair.

For each active object D (device or CPU), we define *access*[t,D] as the set of accesses (object, access type pairs) that object D makes at time t. We also define *owner*[t,D] as the user on whose behalf the device is performing accesses at time t. For example, the CPU always acts for some given process. Then protection is enforced if $\forall t\ \forall D$ *access*[t,D] is a subset of *accessible*[t,*owner*[t,D]] We say that a system is secure if

(1) accessible sets describe the desired security policy and
(2) protection is enforced.

The preceding is a simple conception of security. Nevertheless, it can yield meaningful protection and a comfortable user environment as we demonstrate below. Such questions as whether user A can steal user B's files are, for example, directly handled by this model. Note that point one above contains aspects which will not be mathematically verifiable, since what is involved is a mapping to precise specifications from the intuitive notions which express user desires. One can check internal *consistency* of the resulting specifications, however.

THE CONCEPT OF AN OPERATING SYSTEM SECURITY KERNEL

The value of segregating operating systems code into more or less disjoint layers has been clear for some time. One interesting question that presents itself concerns what the lowest level of the operating system ought to contain. Designs focussed around message handling, process primitives, and others have been suggested. Recent efforts concerning the "nucleus" of the operating system are illustrated by Brinch Hansen[5] and Wulf.[4] There is still considerable debate over this question, and its importance is not likely to decrease as systems grow more complex and additional layers, such as more sophisticated data management code, are added.

It is our contention that the very lowest system level, the subnucleus, ought to contain the security relevant portions of operating systems primitives, and nothing else. The subnucleus that contains this isolated code we call the security kernel of the operating system. Note that such a design differs widely from current practices, as illustrated by OS/360 in which that code is strewn throughout the operating system.

It has been pointed out many times that security is a crucial, basic quality of today's complex systems. The value of carefully designed modularity in the structure of operating system code is also clearly recognized. These two thoughts conjoin; *enforcement* of the intended modularity by security controls is of great aid in developing and understanding the system software. It is easier to obtain reliable enforcement of modularity if the proper functioning of those mechanisms which provide that enforcement does not depend at all on other code. That is, modularity enforcement code should depend on as few and as simple modules as possible. If security is basic to a system, then the code which provides security should also be basic, with few dependencies, hence at the lowest levels of an operating system.

It is exceedingly difficult to understand the interactions of security relevant software when the code and the implications of the design decisions which produced that code are distributed throughout the operating system. The difficulties arising from this distribution are illustrated by the large number of known security flaws which involve the interaction of a number of characteristics of an operating system design and implementation. As one simple example, in a number of systems, the code which checks a security condition and the code which takes the action guarded by that check are separated by some distance in the execution sequence. It is possible under certain conditions, an interrupt perhaps, for the user to change the parameters examined by the check before the action occurs. Such flaws usually depend on the availability of parameters to user code and the interruptibility of the check-action pair.[6] As the subleties of these interactions are seen, the near hopeless quality of poorly planned and distributed security relevant code becomes more and more clear.

The isolation and centralization of security relevant code provides more than merely a better basis for understanding that code, important as that understanding is. By isolating that code at the heart of a system, running on the bare hardware, its correct functioning does not depend on the proper behavior of other modules of operating system software. This fact provides several advantages. Outer layers of the software may be written and modified without need to reevaluate the security of the entire system. Maintenance is then eased. In addition, the resulting security kernel as a relatively small, isolated, independent set of programs is susceptible to a *formal verification of its correctness* with respect to security. The importance of this property is difficult to over-emphasize.

A note of caution is important here, however. It has been suggested that once the behavior of the security kernel has been verified, the security of the entire operating system software has been guaranteed. Such a statement is not necessarily true. If one's concept of the kernel includes only security *primitives*, then it is quite conceivable that those primitives could be inappropriately applied at an outer level, allowing relatively subtle interactions of operating system features that lead to security flaws.

As an example, in the bulk input of a well-known time sharing system, I/O is handled by a separate process. A card deck, along with the desired file name and the target directory, is effectively given to the I/O process which loads the deck into a file and places an entry in the target directory. To do this, the I/O process has powerful access privileges. By this means, anyone may place an arbitrarily named file in any directory.

When a user executes a system command, his directory is searched before the public directory for that command name to obtain the code to execute. The interaction of this search order with the power of the I/O process results in security failure. To exercise this flaw, one merely prepares a program that performs an arbitrary action, and has it loaded into the directory of a highly privileged user. If the program is given a common command name (like SYSTAT or DIRECTORY, for example) then that arbitrary program will be inadvertently executed, with the capabilities of the privileged user, when he types the command. With care, no one will even know the error occurred. To indicate the subtlety of such flaws, it should be noted that if the search order used by the monitor were reversed, that is, if system directories were searched before user directories, this particular flaw would vanish.

In the UCLA work, this class of problems has been examined closely, and solutions to it are suggested below in the sections on virtual machines and verification. As a result, we are convinced that the kernel concept can yield certified security if it is carefully employed.

Another effective argument in favor of the security kernel approach results from the fact that it has been possible to separate security relevant code from the rest of the operating system software and centralize that code. While the verification of the UCLA system was not complete when this paper was written, the kernel isolation design had been finished for some time.

Exactly what capabilities belong in the kernel is perhaps best explained by detailed examples, but several general remarks can be made. First, the distinction of Wulf[4] between mechanism and policy is useful. For example, while the policy contained in the scheduling process can and should be excluded from the kernel, the code which serves as the mechanism to load a process must be part of the kernel. Otherwise, parts of several processes could be loaded together (for example, the general accumulators from process i and the memory management registers of process j).

Second, the kernel can be allowed to make calls to outer layers of the system as long as the proper behavior of the kernel does not depend on those outer calls. For example, the scheduler may be called by the kernel to determine which process to run next. What reply the kernel receives is irrelevant to protection questions. The hierarchical structure is necessary for proof dependencies only, not flow of control or other behavior. This ability to allow certain outward calls is one means by which nonprotection issues can be largely excluded from the kernel code. Of course, as noted by Wulf,[4] *when* the kernel calls the scheduler is itself a scheduling decision; thus this and other questions have not been completely excluded.

VIRTUAL MACHINES AND SECURITY

The concepts of virtual machine designs, in the sense of CP-67 or VM370, have grown in popularity and importance recently. One can view the virtual machine monitor (also called a control program, or hypervisor) as providing some basic functions of the traditional operating system, such as separation of processes, device scheduling and processor allocation. It does not enrich the process environment by providing user services. Instead, the environment produced, called a virtual machine, is essentially logically identical to the bare machine. For discussions of the value of virtual machines to operating systems design and program transferability as well as construction details, see References 7, 8 and 9.

Virtual machine designs have significant advantages in multiuser computer security systems, however, apart from the values mentioned above. One of these advantages accrues from a practical question—the amount of work required to produce a secure multiuser system. Earlier, the desirability of a security kernel was discussed. The presence of kernel code, however, changes the environment that any program sees when it is run. Operations such as changing relocation register values and direct execution of I/O may not be performed, and attempts to do so either will be impossible (when relevant areas are excluded from an address space, for example) or will abort. One is thus faced with either designing or modifying all programs to run properly with the kernel, or providing layers of code over the kernel in order to construct a more suitable program interface. The latter option could imply the building of an entire operating system.

Instead, one might layer over the kernel a skeletal virtual machine monitor (VMM). Such a task is simpler than that of building an operating system. The VMM contains no user services, and its code is devoted in large part simply to simulating those portions of the bare machine which have been usurped by the presence of the VMM itself. An elementary scheduler is of course necessary, and careful attention must be paid to I/O. Nevertheless, a VMM is still much simpler than an operating system, as might be illustrated by the relative sizes of CP-67 and OS/360. The UCLA PDP-11/45 contains certain hardware modifications to reduce the amount of supporting code for the VMM in the kernel. See the appendix for details.

Typically, user services on a virtual machine system are obtained by running a standard operating system in one of the virtual machine environments. Thus, construction of a VMM is a relatively simple and cheap way to obtain a clean interface for programs, and the amount of effort required to demonstrate the utility of the kernel is decreased.

Such an approach involves protecting and controlling entire virtual machines. Hence two users running under an operating system in a single virtual machine are accorded no protection from one another by the security kernel, although they are protected from other virtual machines. The obvious solution to this problem is to run each user with his own operating system in a separate virtual machine, using execute only sharing of common operating system code to maintain efficiency.

A second advantage of a virtual machine approach however is more intrinsic, but also accrues in part from its simplicity. Earlier, it was demonstrated with the example of bulk I/O that, depending on one's conception of the kernel, it was potentially possible for higher levels of software to misuse kernel primitives in a fashion that could lead to a security flaw. This problem is diminished if the higher levels of software are simpler.

Inter-segment linking, access control over spooling processes, and many other operating system features are absent in a virtual machine monitor. The complexity of the "protection semantics" associated with aspects of operating systems such as those mentioned above, as well as the necessary supporting code, can easily lead to security errors. By contrast, a virtual machine system projects a very simple environment from the point of view of security. Sharing may be provided by very simple methods while still maintaining the richness of an individual user's environment.

This relative simplicity makes it practical to demonstrate that there do not exist interactions of the sort mentioned earlier which cause security flaws. Essentially, there are many fewer features to interact.

COMMUNICATION CHANNELS

Another relevant view of security deals with the control of communication. For example, a system for which a user with one clearance could not pass certain data to a user with a lower clearance no matter how hard both try would be of value to the military. This problem essentially is one of showing that there does not exist a communication path between two processes. Up to this point, we have been considering a fairly specific definition of security, albeit one that can involve fairly subtle interactions of code. However, as pointed out by Lampson,[10] there are a number of conceptual channels of flow of information in a shared system. First, there is the obvious one that most people speak of in the context of security, the explicit passing of files and other units of information, reading another user's work space, and the like.

But there are others, often involving the passing of resources—assigning and deassigning devices, or making service demands on the system that other users can sense. Exactly what channels exist, what their respective potential bandwidths are, and what mechanisms and costs are needed to seal them are all questions that need to be resolved in order to decide how these communications channels should be handled.

Of these channels, we first distinguish between those which depend on timing considerations, and those which do not. A timing dependent case is demonstrated by process A sensing the CPU load of the system caused by process B. In the context of a virtual machine system, one way these channels may be blocked, at least in so far as the running programs are concerned, is by properly simulating the passage of *virtual* time to VMs, and not providing a real time measure.

Timing independent channels also exist, however. Consider the following case. Let P1 and P2 be separate processes who are not to communicate. The devices D1 and D2 are discs controlled by a common scheduler S. Requests for I/O may be considered as messages from P1 and P2 to S, and completions as messages from S to P1 and P2. Notice that in this example, both P1 and P2 are using both devices.

In keeping with the kernel design philosophy, it is desired that S be kept out of the kernel. How then are P1 and P2 prevented from communicating via the scheduler S? Both send S information, and both receive information from S. Are we forced to prove assertions about S? If our system already existed, its design were as above and we were attempting to secure it in retrospect, without major redesign, the answer is likely to be yes.

Instead, let the kernel do all the moving of information, and treat the messages as being contained in "locked boxes." A scheduler may *read* (*but not change*) such a locked box. Label messages from P1 and P2 with their destination, D1 or D2. The return messages from devices are similarly labelled from either D1 or D2. The scheduler now merely queues requests and hence cannot change message contents.

SOURCES, SINKS, AND THE MORSE CODE PROBLEM

The above design does not completely block interprocess communication, for a "Morse Code" mechanism still remains. That is, both P1 and P2 can pass S two types of tokens that S can distinguish: D1 type and D2 type. Hence S can receive an arbitrary binary encoded communication from each of the two processes. Furthermore, S can send a nearly arbitrary message back to each process by ordering the return of completions (*nearly* arbitrary only because he may run out of tokens of one or the other type or both). We have been had by a malevolent scheduler. Skeptics should remember that binary codes are the charwomen of contemporary computing.

This problem however is not intrinsic to security systems nor an inherent defect in the kernel concept, but rather merely one of inappropriate design. Split the device scheduler S into two schedulers, S1 and S2, one for each device. Now each scheduler, if we continue to operate in "locked box" mode, deals in only one variety of token, and hence there is no binary communication channel.

The preceding is a rather attractive solution. The schedulers S1 and S2 need *not* be proven at all, and they may have full access to information necessary for scheduling. Little compromise with respect to performance has been made. The two schedulers may even share common code, as long as their writable storage is separated. The task of providing proven security has been eased, with the only important cost having been careful design.

The above solution is valid because S1 and S2 are purely information *sinks*. While they receive message contents, there is no way for them to broadcast, so long as only one request per process per scheduler may be pending at any time. They only request that the kernel return device status to the processes.

More generally, the solution is effective because the flow of information with respect to S1 and S2 is *one way*. In this case, the two are sinks. It would have been equally sufficient with respect to security if they had been pure *sources*. Of course in this case that would not provide the desired functionality. The principle here that does generalize usefully is that one can reduce the security problem significantly by first isolating as much as possible, often in ways that are not immediately apparent. This principle has been applied to the UCLA kernel a number of times, the device schedulers and virtual machine simulators being two examples.

At this point one might argue that these subtle channels are in certain respects irrelevant because they seem to require the active, coordinated cooperation of all the parties involved in the communication. If two users wish to communicate, let them; isn't that the strength of multiuser systems anyway? However, in the disc scheduling example raised earlier, even without process P1's cooperation, it would have been possible for P2 to learn about P1's I/O characteristics. More important, security may be compromised by incorrect high level design. Errors in outer modules, such as the scheduler, are exploitable. Careful proof procedures should discover these cases.

VERIFICATION

It is important to realize that even if no security verification were actually performed, the design philosophy described here and the discipline imposed by the intent to verify, together, have already increased the reliability of the security aspects of the system. Nevertheless, there is more which can profitably be done; verification is an attainable goal.

Program verification tools are not yet in a well developed state. The largest program that is known to the authors to have been verified to date contains somewhat more than two thousand instructions; small compared to most operating systems.[11] Hence, it is necessary to severely limit the size of the program about which one wishes to prove certain properties. The UCLA kernel is approximately 500 lines of high level code. We are in the process of verifying that code and expect to be able to complete this task with a reasonable amount of effort.

It is important to realize that program verification is not the whole of operating system security, although it is a very important part. Verification establishes the *consistency* of a program with a separate, essentially static statement of what that program is, or is not, to do. In order to apply verification, one first needs explicit definitions of security, and those definitions must be translated, currently by hand, into predicates that may be handled by mathematical means. Second, one is faced not by a large flowchart, but rather by a number of kernel primitives, which potentially may be invoked in any order. In addition to verifying certain properties of the primitives themselves, it is necessary to demonstrate that there is no order of invocation of primitives that would result in a security violation. If the primitives are thought of as composing an action space, then one needs to demonstrate that there does not exist a path through that space, the result of which would make one of the security predicates not true.

There are a number of strategies which can make this proof process easier. After carefully categorizing every independently addressible object and action to make the model complete, naming techniques can be employed to segment the proof task. If we can demonstrate that objects only have a user's name associated with them if they are members of his accessible set, then we can show protection enforcement by showing he can only access objects with his name.

As a second proof strategy, certain required predicates may be included as run time checks. For example, in a military system, one might wish to guarantee that a user with secret clearance is not able to access a top-secret file. One way to guarantee that constraint is to embed a run-time check in the (only) I/O routine.

All of these considerations help to make the verification task possible although far from easy. They also support the undesirability of ex post facto verification.

THE UCLA-VM SYSTEM

Let us give some examples of the previous remarks from the UCLA-VM system. A sketch of the structure of the UCLA virtual machine system is shown in Figure 1, with the VMM broken into its attendant parts. The objects in the system have been specifically defined and are homogeneously treated and viewed by the kernel. That is, the kernel has no knowledge of the internal structure of any security objects with the exception of the protection data. This data is the basis for security decisions, and plays a role analogous to the contents of the Lampson-Graham-Denning protection matrix.[12] It is packaged into security objects so that access to the data may be controlled. Thus, control is obtained over the way protection decisions are changed. The blindness of the kernel to other objects' internal structure simplifies matters, but it implies that only actions among objects are monitored; no intra-object control is provided. As a result, activity within a virtual machine is not controlled by the kernel. Two users running in a single virtual machine are accorded no protection from one another by the kernel.

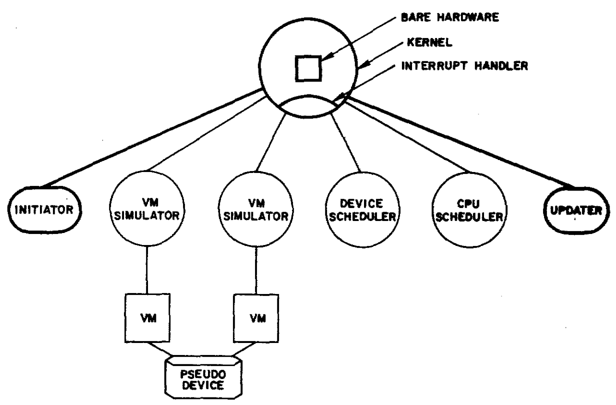


Figure 1—Virtual machine design

In order to be protected, they must each run in a separate virtual machine.

The modularity shown in the figure is needed for security. Each VM has an associated simulator. This code performs the task of simulating the virtual machine environment. However, these simulators are not proven correct. Thus to assure security of each user's data, each simulator must be logically separate with no shared writable storage. For similar reasons, and to avoid Morse code problems (as discussed earlier), the CPU scheduler and all shared device schedulers must be logically separate.

However, to be of practical utility, some sharing among virtual machines is necessary. For example, one virtual machine contains the only ARPA network[13] interface software, and it is highly desirable that the network be available to other virtual machines. Hence, a shared read/write segment facility between two virtual machines is included. The simulators for two virtual machines may share such a segment if so indicated by the protection data. Pseudo device interfaces to the shared segments are also provided so that standard operating systems may communicate with one another without other special facilities.

Thus, the design which is being verified supports only limited sharing, via shared read/write segments and pseudo devices. As a result, the system is rather simple and the semantics of security fairly straightforward. This simplicity has been a substantial aid to the system's development. In addition, shared execute-only segments are also supported, so that multiple users running in multiple virtual machines but executing under separate incarnations of the same operating system can use the same code.

In Figure 1, portions of the system's structure are highlighted. It is these portions upon which the security of the system depends. The kernel of course is included, but two modules outside of the kernel are too. The initiator and updater perform authentication as part of their primary functions. A user first presents himself through a terminal to the initiator, who eventually passes him to a virtual machine.

The updater's task concerns the changing of protection data. It is only through this port that access to the data is potentially possible. This procedure is necessary since the state of the protection data must be guaranteed in order for the rest of the mechanisms discussed here to be meaningful. One needs a reliable channel to the kernel which does not pass through unverified VM-simulator and operating system code in order to inspect and change that data with confidence.

The UCLA-VM System is useful for practical work besides being a testbed for the development of techniques for constructing secure operating system software. The PDP-11/45 serves as an entry way to the ARPANET, replacing older equipment. The network interface software is provided by ANTS (ARPA Network Terminal System), which expects a bare machine on which to run. In addition, it will now be possible for the processing power of the 11/45 to be available for network measurements[14] as well as local computing, not allowed by ANTS. The virtual machine monitor provides the ability to concurrently run a number of applications, each of which logically expects a bare machine, and also yields a good environment for instructional purposes, especially for "hands on" operating systems experience.

COSTS OF SECURITY

The costs of providing multiuser computer security are incurred in at least several ways: construction, user convenience, and performance. Let us consider each of these in the concext of a UCLA-like system.

The cost of construction of the verified kernel with simple, understood semantic layers above it (the VMM), is certainly reasonable. The project will have consumed a small number of man years of high quality effort and no unusual equipment. A significant amount of this work would not be required if the task were repeated for another machine, and much of it is a basis for extensions.

In terms of convenience, while the system is not yet operational, it is clear from other systems that a virtual machine environment is a comfortable one for many users. The simple sharing mechanisms provide a necessary basis for network communication and inter-machine interaction.

With regard to performance, a definite answer will not be available until operational tests are made. Nevertheless, a number of remarks can be made. A definite upper bound on the performance cost of security can be obtained by comparing the performance of a procedure under the virtual machine monitor with its performance on the bare machine. Of course, that bound will also include the overhead of the virtualization process as well as limitations imposed by security.

It has been suggested that the cost of security in general may be expensive. This expense will result from overhead imposed by the necessity to follow procedures which, without security considerations, could have been obviated. Indeed, we have found places in which the procedures to perform a desired task are considerably more lengthy than they would normally be, I/O to a shared device being a case in point.

Nevertheless, we expect our security degradation bound will demonstrate that the fears of inefficiency are incorrect.

We will also estimate what portion of the observed overhead is due to virtualization costs. Hence, we confidently expect to demonstrate that a simple, but useful form of verified multiuser security may be obtained at a quite acceptable cost.

CONCLUSIONS

This work has been intended to demonstrate several points. First, and perhaps most important, *it is practicable to have verified software security in multiuser computing systems.* Second, the approaches of kernel design, virtual machine monitors, and mathematical verification of the properties of software contribute usefully to the task of providing verifiable security.

Nevertheless, a great deal remains to be done. It has been and continues to be a taxing effort to obtain this high level of software security for any given system, in part because our tools and concepts are still unrefined. These facts are encouraging, for they suggest that considerable progress can be expected.

Furthermore, the system described here provides only limited sharing, and does not, for example, address the problems of mutually suspicious subsystems or memorylessness at all. It has been argued that multiprogramming of resources, rather than information, is still the predominant activity, at least among the security conscious segment of the computing community. Although this segment would be satisfied by a virtual machine approach, there remain vital activities for which reliable control of sharing is crucial, and those activities are not expected to decrease in the future.

APPENDIX

The security kernel and virtual machine monitor are being constructed for a DEC PDP-11/45 that is being attached to the ARPANET at UCLA. The PDP-11/45 has a three state architecture which naturally lends itself to the needs of the kernel, VMM, and user environments. However, the Unibus I/O structure of this machine does not lend itself conveniently to virtualization, since nearly every instruction is potentially an I/O instruction and most be simulatable, unlike the limited set that normally accompanies machines with more conventional I/O processors.

However, much more important than this inconvenience with respect to I/O is the fact that the standard PDP-11/45 cannot be virtualized at all. In Reference 8, it is stated that hardware must have certain characteristics in order for a VMM to be constructible. Briefly, instructions that affect control of the processor, or whose behavior are disturbed by the presence of the VMM, are termed sensitive instructions. All sensitive instructions must be privileged in order that they may trap to the VMM to have their effect simulated. In the standard PDP-11/45, there are nine instructions for which trapping is necessary but which are not privileged. This fact makes the machine impossible to virtualize.

In addition, one would like trapping of instructions to be a function of the mode in which attempted execution of the instruction occurred. The reason such behavior is desirable is a result of the following considerations. It will be natural to run the kernel in the most privileged mode, the virtual machine monitor in the next most privileged mode, and the virtual machines in least privileged mode. One then prefers that instructions which trap in a virtual machine be reflected by the hardware to the VMM, while it may be necessary that the same instruction executed by the VMM trap to the kernel. Such mode dependent trapping has been suggested before.[15]

In the case of recursive virtualization, in CP-67 for instance, this behavior is simulated by the software. The hardware traps all instructions to privileged mode, and software reflects some of them out. Here, however, there is an additional motivation. The existence of mode dependent trapping makes it unnecessary to have the reflection software in the kernel. It needn't exist at all. As emphasized in the body of this paper, the need to exclude non-security code from the kernel is almost as important as including all the relevant code. The DEC/UCLA hardware modification package also includes other features for efficiency and/or convenience.

REFERENCES

1. Schell, R., Private communication, USAF/ESD L. G. Hanscom Field, September 1972.
2. Lampson, B. W., "Dynamic Protection Structures," *Proceedings of FJCC*, 1969.
3. Saltzer, J., "Protection and Control of Information Sharing in Multics," *ACM/SIGOPS Symposium on Operating System Principles*, Yorktown Heights, October 1973.
4. Wulf, W., *HYDRA: The Kernel of a Multiprocessor Operating System*, Carnegie-Mellon University, 1973.
5. Brinch Hansen, P., "The Nucleus of a Multiprogramming System," *CACM*, April 1970.
6. Bisbey, R., Private Communication, USC/Information Sciences Institute, June 1973.
7. Buzen, J. P. and U. Gagliaridi, "The Evolution of Virtual Machine Architecture," *AFIPS Conference Proceedings*, Volume 42.
8. Popek, G. J. and R. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures," *Communications of the ACM*, Vol. 17, No. 7, July 1974.
9. [IBM CP-67] IBM Corporation, *Control Program-67/Cambridge Monitor System*, IBM Type III release No. 360D-05.2.005, IBM Program Information Department, Hawthorne, New York.
10. Lampson, B. W., "Dynamic Protection Structures," *AFIPS Conference Proceedings*, Volume 35.
11. Ragland, Larry C., *A Verified Program Verifier*, Ph.D. Thesis, University of Texas, Austin, 1973.
12. Graham, G. S. and P. J. Denning, "Protection—Principles and Practice," *AFIPS Conference Proceedings*, Volume 40.
13. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings*, Volume 36.
14. Kleinrock, L. and W. E. Naylor, On Measured Behavior of the ARPA Network, to be published.
15. Goldberg, R. P., "Architecture of Virtual Machines," *AFIPS Conference Proceedings*, Volume 42.

An interactive software engineering tool for memory management and user program evaluation*

by WOLFGANG W. MILLBRANDT and JUAN RODRIGUEZ-ROSELL

Brown University
Providence, Rhode Island

INTRODUCTION

As the use of virtual memory becomes more and more accepted, the problem of effective storage management becomes more and more important. To date most efforts to optimize the use of memory have been directed at devising memory management strategies at the operating system level that minimize the number of page faults. For example, Comeau[1] has shown that the loading sequence of subroutines can have a considerable effect on paging activity. Hence page-fault-optimizing loaders, linkage editors and compilers have been proposed. Although the concepts of "locality" and "working set" have been known for some time (c.f. Denning[4]), little effort has been made to provide the programmer with suitable tools for making his programs "more local". This seems to stem from the fact that, short of notions of "modular coding", little is known about what sorts of programming habits actually result in local code. Consequently, most optimization techniques used to date have assumed that user programs were an unmodifiable input to the operating system.

Techniques for increasing locality of user programs and thereby reducing the paging overhead in a virtual memory environment have been investigated by Comeau,[1] Hatfield[7] and Ferrari.[5] The methods presented involve the automatic restructuring of program modules into relocatable segments to increase the likelihood that page references that are close in time will also be close in space. In the Hatfield study the use of online displays to determine optimal segment sizes and view the effects of program reordering was found to be exceedingly helpful, but use of their system as an everyday programmer feedback and user program monitoring tool was never fully exploited.

The Brown University Display for Working Set References (affectionately known as BUDWSR) was primarily developed as a user-oriented tool to fill the need for user feedback systems by enabling the programmer to interactively monitor the memory referencing behavior of his modules. It was hoped that the programmer would be able to get a "feeling" of what it means to write localized code, and hence be able to modify his programming techniques in order to reduce, or at least have more control over, the memory resources required by his program. Although BUDWSR has not been in existence long enough for us to evaluate its effectiveness as a programmer training or feedback tool, we have already been able to establish some simple guidelines that allow the programmer to measure and hence increase the memory utilization of his programs. These guidelines could be used as a basis for establishing engineering standards for program evaluation. Furthermore, BUDWSR has proven itself to be an extremely powerful systems programmer tool that greatly facilitates the manual repackaging of modules in order to increase memory utilization.

SYSTEM ORGANIZATION

The system (c.f. Figure 1) essentially consists of a System/360 machine language interpreter and a satellite display processor. The user runs his program in much the same way as he would in a normal CP/CMS[2] environment except that the interpreter counts the references made to memory pages (the page size is user defined) and periodically transfers its tables to the satellite computer. The interpreter runs in a 512K byte virtual machine and simulates a 256K CMS environment for the user's program; the data gathered thus reflects only the activity of the user's program and is not dependent on the virtual machine's external environment. Since the System/360 instruction set is highly formatted, the basic data gathering facilities of the interpreter are fairly simple* and represent an acceptable cost

* This work is sponsored in part by the National Science Foundation, grant GJ-28401X, the Office of Naval Research, contract N00014-67-A-0191-0023, and the Brown University Division of Applied Mathematics.

* Most RR format instructions can be executed directly via the System/360's execute instruction, while most RX, RS, and SI format instructions can simply be executed after computation of the base displacement address. Optionally, BUDWSR will not perform complete interrupt processing, e.g., interpretation stops when an interrupt is generated by the instruction stream, and resumes when control is returned from the interrupt handler. Since many of the CMS nucleus pages are in shared care, references to the nucleus code should not result in any significant paging overhead to the user.

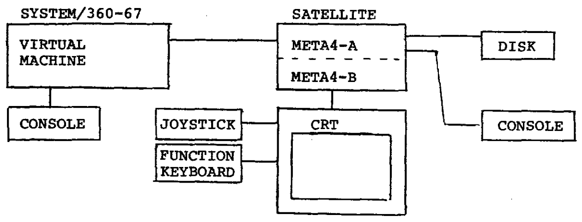


Figure 1

(roughly 50 times the cost of the original program) for gathering memory utilization statistics.

The basic display (c.f. Figure 2a) produced by the satellite consists of page addresses plotted against time (measured in number of instructions). For each page address and time interval the system plots a small vertical spike whose height indicates how often that page was referenced during that time interval. At the start of the trace the user may define an arbitrary page size (from 8 bytes to 128K bytes) and time interval (from 1 to 65,535 instructions). Furthermore, he may ask the satellite to perform a non-destructive compression of the reference data, e.g., the user may ask the interpreter to gather statistics for 128 byte pages, and then view the results for 128, 256, . . ., or 128K byte pages. This allows him to view the programs behavior both on a global level (using for example a 4K byte page size) and to selectively "zoom-in" on any local peculiarities. To aid the programmer in identifying and restructuring inefficient modules, a cumulative reference count for each page, and module names and entry points (obtained from the CMS loader tables) are also displayed.

The satellite part of the system also uses the page reference data to produce a graph of the working set size versus time (c.f. Figure 2b). Thus the user not only has available which specific memory pages were referenced but also a summary of the total memory used during a time interval. The working set size is computed in bytes for the user specified page size and also for the System/360-67 page size (4K). With a small page size (typically 128 bytes) the difference in the working set sizes indicates the amount of wasted space in the System/360-67 working set. Currently, we have measured only working set sizes and re-entry rates for an individual process. In reality, the amount of memory allocated to a process is a function of the paging algorithm and the load of the system. (Some data on system page fault and re-entry rates can be found in Rodriguez-Rosell and Dupuy.[9]) We have not concerned ourselves with simulations to determine page fault rates as a function of memory size or the operating system's memory management policy. In general the amount of real memory available to the user is beyond his control; however, by using BUDWSR, the programmer can improve the locality of his programs while developing them, conceivably even changing to algorithms or data structures that might prove less noxious to the system. In return, he hopes that if his process uses memory efficiently the operating system will allow his process to execute rapidly.

The use of the satellite processor[10] came about quite naturally in that we approached program monitoring as two

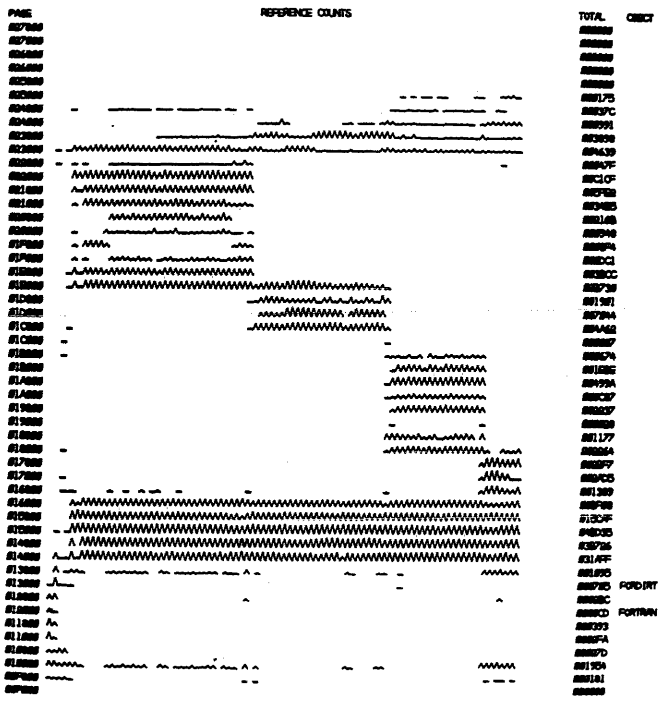


Figure 2a

separate tasks, data gathering and analysis. We also wished to be able to monitor the output of the tracing program while it was running, rather than wait for off-line hard copy output of the execution record of the monitored program. Thus the satellite performs the multiple tasks of communication with the System/360 interpreter, updating the disk-resident master file containing the page references, and updating the display itself as new data are transfered by the interpreter or when the user "scrolls"* through, or locally alters the page size for the memory display. The use of the satellite to process the trace data makes the system extremely flexible. Thus whenever we wish to determine whether or not a specific memory utilization parameter is meaningful, it is a simple matter to add code to the satellite program in order to calculate and display the parameter.

Another effect of using a satellite processor rather than a process running on the virtual machine itself is the fact that the interpretation and display data handling proceed in parallel in "real time." The interpreter runs unmolested by user interrupts for larger page sizes or different display lists. Thus it is possible to view a trace of a system, repackage it, and/or retrace it with different parameters all in the span of a few minutes. For monitoring an interactive system,** we can view a trace at the same time we are specifying commands to that system in order to quickly isolate and reduce the memory requirements for inefficient command modules. Since the trace data are immediately available after each time interval, it is also unnecessary to correlate the data from an entire run (such as would be given by an off-line plot) to the specific command or sequence of commands that generated the data. It is of course also possible to view previously taken trace data in a stand-alone mode of the satellite.

PARAMETERS FOR EVALUATION FOR PROGRAM MEMORY UTILIZATION

To date the system has demonstrated its usefulness through several practical applications. Many of the facts noticed about program behavior could of course have been predicted by use of common sense, and the reader may thus find some of the results rather unstartling. However, the fact that most of the observed traits are found in a large class of programs indicates that common sense rules (such as presented in the next section) are not used or even fully understood. It is hoped that by providing effective systems measurement tools and developing memory utilization standards, programmers will be made aware of their bad techniques (and also their good ones). Furthermore it has been found that a display is quite effective in pointing out some of the parameters that may be used to define standards

* In general, the information to be displayed is larger than the available display area (e.g., the CRT display can effectively represent 50 4K-byte pages, but not 800 256-byte pages).

** For example, some of this paper was edited under a text editor being monitored (see below).

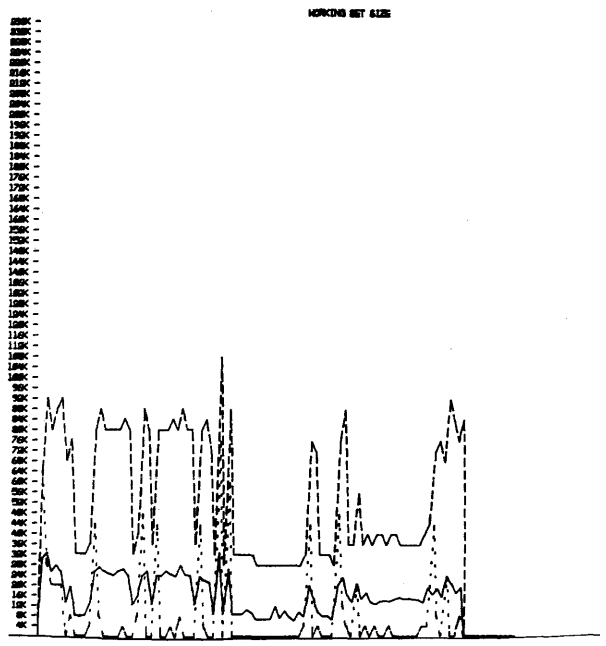


Figure 2b

for efficient memory utilization. The following parameters have been found to be particularly useful:

Working set size—the number of pages that are referenced during a time interval, times the page size (hence the working set size is expressed in total number of bytes that are demanded by the program). In general, when we refer to a programs working set size, we are referring to its mean size over all the intervals.

Memory utilization—ratio of memory used based on the user specified page size as compared to the total amount of memory allocated based on the 4K page size of the System/360-67. With a small user page size (typically 128 bytes) this ratio approaches the percentage of the total allocated memory actually used by the program. It should be noted that although working set sizes vary significantly from program to program, the memory utilization factor consistently ranged between the rather low values of 30 to 50 percent for most of the programs observed. Generally programs having the higher utilization factor had been optimized with respect to instruction stream execution, with most of the remaining inefficiencies a result of poor *data* referencing characteristics.

Working set entry rate—this parameter indicates the amount of the current working set that was not present in the previous working set. It is useful to separate this parameter into two components. The first, which we will attribute to locality unstability, is a significant change in the size of the working set from one time interval to the next. The working set size will temporarily increase as a program moves from one locality to another and pages from both localities remain in the working set (see, for example, the trace of the FORTRAN F Compiler in Figures 2a and b). Consistently unstable working sets are indicative of poor process behavior.

The second component of the working set entry rate, we will attribute to page replacement rate. It is a measure of the change in the membership of the working set as a result of single pages entering or leaving the working set. Thus programs with a nearly constant working set size may still have a non-zero re-entry rate due to page replacement within the working set. This component of the re-entry rate was found to be meaningful only for programs with large ($>$40K) working sets. Program references to infrequently used data areas or error routines seem to result in a small set of "fringe" pages that wander in and out of an otherwise stable working set. Little if any page replacement was observed for programs with small working sets.

It should be noted that individual parameters are quite sensitive to the time interval used in collecting the data. Thus it is quite possible to use a small time interval to artificially reduce the size of the working set. This is however offset by the fact that if a program is using memory inefficiently, reducing the time interval will correspondingly increase the re-entry rate of new pages into the working set. Conversely, although a long time interval will increase the size of the working set, it will also decrease the re-entry rate. The time dependencies of the parameters can be used quite advantageously in the process of packaging system modules. Thus we can first choose a long time interval and package to reduce the total size of the working set as much as possible. We can then use a small time interval to repackage the heavily used routines found by the first packaging into a stable working set configuration. After the initial packaging iterations it is still possible to fine tune a user system by appropriate time interval selection. Thus if we are optimizing an interactive text editor we might select a time interval that spans a single editing function; if we are optimizing a CPU bound applications program we might choose a time interval that corresponds to the operating system's time slice, etc.

SOME CASE STUDIES

FRESS—A file retrieval and editing system

FRESS[6] is a sophisticated interactive text handling system. It has run in a 120K MVT partition (using dynamic loading), but in the CP-67 environment it was found more convenient to relinquish the memory management to the virtual paging system. Even though FRESS is a highly modular system, and entirely code in assembly language, it was designed and implemented without paging load criteria in mind. Thus it was not too surprising that the FRESS users were typically the first to be impacted at times when Brown's 67 neared saturation.

The following observations of the unpackaged (e.g., as loaded by the CMS loader) version generally accounted for this behavior:

- The working set size was rather large, roughly 128K (with a 4K page size and a time interval of 5,000 instructions).
- The working set was extremely unstable, i.e., the working set size would vary from 60 to 128K between any two user activated functions.
- And finally, the amount of the CP-67 working set actually used during any one time interval was typically around 30 percent.

Fortunately it was possible to remedy most of the above characteristics by simply reloading the system. Thus after noting which modules were frequently in use, and which were not,* and hand specifying the module loading order, the following results were obtained:

- The working set size dropped to 80K (a reduction of 30 percent)
- The working set size became more stable, generally changing by less than 16K for the common editing functions.
- The page utilization figure increased to 50 percent.

Even with the increased memory utilization achieved by a repackaging of the FRESS system, further reductions of another 20K in the size of the working set still seem possible by breaking up some of the larger modules and alignment of some data areas on 4K page boundaries (some guidelines to enhance the "packageability" of programs are presented in the next section).

The CSS editor

The CSS editor[3] is a line oriented text editor that is provided as a user service on the NCSS time sharing service. Since NCSS's Duplex 67 must frequently support many users the following statistics should not be too surprising:

- The working set size was a small 12K$\pm$4K (with a 4K page size).
- The working set was very stable due to the fact that most of the commonly used functions had been loaded onto the first CMS user memory page.
- The memory utilization fluctuated from 30 to 50 percent depending on the user function. The lower memory utilization figures were usually due to editing functions that generated data references across the 4K page boundaries. Since many fixed length line data files contain about 40 percent blanks, one wonders how much the utilization factor would increase if some data compression were built into the editor.

* In particular some modules, such as the command language interpreter, are executed for every command; some, such as the various editing functions are seldom invoked at the same time, but do share some common subroutines, etc.

SCRIPT AND NSCRIPT

SCRIPT and NSCRIPT are text formatting programs that can be used in conjunction with the CMS or CSS text editors. NSCRIPT is an MIT version of SCRIPT that supports some more advanced features such as footnote placement and user macros. For purposes of comparison we monitored both programs while they processed the *same* input file (obviously none of the extra features provided by NSCRIPT were included in the file). We let the user draw his own conclusions from a comparison of the following data:

- The SCRIPT working set size was 8K.
- The working set was stable.
- Memory utilization ranged from 50 to 70 percent.

- The NSCRIPT working set size was 24K.
- The working set was stable.
- Memory utilization ranged from 30 to 50 percent.

FORTRAN F Compiler

The Figures 2a and b are representative of any particular Fortran compilation, e.g., changing the source program to be compiled may change the duration of each phase but will not significantly change the characteristics of the two graphs (here we are assuming that no exceptional conditions such as error messages occur). The page size for Figure 2a is 2K bytes (for a more detailed description of the figures see the first part of this paper). The plots in Figure 2b are working set size with a 4K page size (top line), working set size with a 128 byte page size (center line), and entry rate for 4K pages (bottom line). The time interval for both figures is 10,000 instructions.

Some CPU simulation programs

One of the earlier demonstrations of our system was for some representatives of a systems measurement group at the Mitre Corporation who brought two of their programs for observation on our system. The first was a Fortran program which had been analyzed as to the number and kinds of source statements it contained. The second program (also written in Fortran) accepted the analysis data as input and simulated the "resource usage" behavior of the first program. We were to observe the behavior of both programs to see how good a job the simulator program was doing. Although the CPU load of the two programs could have been considered to be equivalent, we observed with BUDWSR that their memory utilization patterns were in no way related to each other. This might serve as a reminder that program behavior is still poorly understood, and that a good deal of empirical data gathering might still be in order.

SOME PRACTICAL GUIDELINES TO INCREASE MEMORY UTILIZATION

Many of the suggestions in this section can also be found in Morrison.[8] We have presented here those situations that have been observed to have the greatest potential for reducing the memory requirement of a program. Most of these observations are applicable only to the instruction stream of the monitored program, and not to the data references. We are currently experimenting with monitoring only data referencing (read and/or write) patterns in order to be able to understand some of the finer details of memory management. It should also be noted that a good systems programming tool, such as BUDWSR, is an invaluable addition to the commonly used guidelines.

- System modules should be as small as possible since this greatly facilitates global repackaging.
- A module should execute as much of its code as possible when it is called. Hence special cases and error conditions should be *diagnosed* inline, but handled by calls to separate modules.
- Large initialization sections should be handled as calls to separately created modules if necessary. This has the effect of compacting the more frequently used code and also allows the programmer to group together the various initialization sequences. The most common case of inefficient memory usage found by our observations was the first 128 to 512 bytes of initialization code in a module, that were executed only at entry to the system.
- Quite frequently the packaging of subroutine libraries used by higher level languages was not done with virtual memory in mind. Thus the system programmer should be especially cautious of some unnecessary overhead brought about by inappropriate loading of run time routines. For example the following PL/I statements generate a 16K to 24K working set in our CMS environment (note that this figure does not include the pages used by the CMS I/O handler!):

```
TEST: PROC OPTIONS(MAIN);
DCL I BIN FIXED;
DO I=1 TO 10;
  PUT LIST (I);
END;
END TEST;
```

- Careful consideration should be given to user management of his data structures, i.e., it is often preferable to allocate a few large chunks of memory, rather than many small ones, so that data references may be organized in an efficient manner. The same holds true for data organization using the Fortran COMMON statement.
- Heavily used routines and frequently referenced control blocks or I/O buffers should be allocated in an order

such that they do not cross page boundaries unnecessarily.

- Software implemented stacks for a subroutine's save area and local variables may be of considerable use in efficiently organizing data references.
- Error message text and error message handlers should be grouped separately from normal flow of control. Frequently used "prompt" messages should of course be kept separate from the error message modules.
- The use of literal pools at the end of large programs (or more typically at the bottom of the first 4K of a program) should be avoided. Better yet, literal data should be treated as part of the instruction stream and placed as soon as possible after its use.

CONCLUSIONS AND FUTURE WORK

It is clear from our experience that the use of on-line displays and satellite processors as measurement tools is far more flexible than use of batch-oriented measurement systems. Satellites will be increasingly used to monitor and assist both the operating system and user software. From an operating system point of view, it is expected that monitoring processors will be attached to large mainframe CPU's[11] as an integral part of the operating system (as is done with the CDC STAR-100). Thus we hope that as the larger systems become more complex, data *gathering* facilities (such as page reference counts and general paging activity) will be implemented in the microcode of the mainframe CPU and be directly available to the satellite for *processing*, thus bypassing the need for system measurement via interpreters, simulators and the like. Just as today, the mainframe relinquishes I/O operations to the channel, future mainframes may pass on user behavior data (by means of control store) to a satellite that will compute new working set size parameters, while the mainframe processes another user. Furthermore, this arrangement makes the behavior of the mainframe system insensitive to the data analysis complexity, as well as allowing modification of the data analysis by a reprogramming of the satellite.

From a user software point of view, it is expected that some standards (for CPU and memory utilization) will be established for programs in production use. Thus the fact that by a simple repackaging it is frequently possible to increase the memory utilization by 20 to 30 percent may indicate that quantitative rules such as "use 50 percent or more of the memory allocated," could be established. These rules, however, will be of little use if we do not provide adequate measurement tools, such as BUDWSR, that allow the user to monitor his programs.

In view of the recent technological developments, it is very possible that future computer systems will have very large random access memories, possibly of the order of 50 megabytes. A new dimension is then added to the evaluation problem, for then the determining factor will be *data* locality, rather than program execution locality. Programs will fit entirely in memory, but the larger data base oriented systems of the future will certainly need techniques to increase the data locality at all levels in the data base hierarchy. Engineering tools are necessary to evaluate and control solutions to user requirements.

REFERENCES

1. Comeau, L. W., "A Study of user program optimization in a Paging System," *ACM Symposium on Operating System Principles*, Gatlingburg, Tennessee, October 1967.
2. *CP/CMS User's Guide*, Form GH20-0859-2, IBM Corp. Technical Publications Dept.
3. *CSS SCRIPT Reference Manual*, National CSS, Inc., 460 Summer St., Stamford, Conn. 06901.
4. Denning, P. J., "Virtual Memory," *Computing Surveys*, Vol. 2, No. 3, September 1970.
5. Ferrari, D., "The Method of Critical Working Sets for the Automatic Improvement of Program Locality," to be published.
6. *FRESS User's Guide*, Text Systems Inc., 106 Highland Ave., Barrington, R.I.
7. Hatfield, D. J., and J. Gerald, "Program Restructuring for Virtual Memory," *IBM Systems Journal*, Vol. 10, No. 3, 1971.
8. Morrison, J. E., "User Program Performance in Virtual Storage Systems," *IBM Systems Journal*, Vol. 12, No. 3, 1973.
9. Rodriguez-Rosell and J. Dupuy, "The Evaluation of a Time Sharing Page Demand System," *Spring Joint Computer Conference*, 1972.
10. Stabler, G. M., and Andries van Dam, "Intelligent Satellites for Interactive Graphics," *NCC*, 1973.
11. Withington, F. G., *Trends in Computer Technology in the Future —Is Centralization Inevitable?*, A.D. Little, Inc. 1973.

Development and implementation of a medical/management information system at the Harvard Community Health Plan

by NORMA JUSTICE, G. OCTO BARNETT, ROBERT LURIE and WILLIAM CASS

Massachusetts General Hospital and Harvard Community Health Plan
Boston, Massachusetts

The Harvard Community Health Plan (HCHP) is a multi-clinic prepaid group practice currently serving 37,000 greater Boston subscribers. The original clinic in Kenmore Square, Boston opened in October, 1969. A second clinic opened in Cambridge in July, 1973, in order to meet a projected subscriber population of 75,000 by 1977.

DESIGN GOALS

Since the founding of HCHP, representatives of the Laboratory of Computer Science (LCS) at Massachusetts General Hospital and the Harvard Plan have participated in a joint effort to design and implement an optimal computer-based medical records system.

The objectives they accepted were those of any traditional medical record system:

1. to provide a mechanism for the recording, storage, and retrieval of information necessary for patient care;
2. to meet the administrative needs of health care management.

Over the last fifty years, medical practice has changed from the single family physician to the concept of a team of medical care specialists, working in a co-ordinated fashion. Because of the multiple physician/patient relationships this implies, the medical record has come to occupy a key role in care delivery. This is particularly true in a prepaid group practice where there is strong emphasis on health maintenance and continuity of care, and where medical services are rendered by a number of specialized health care professionals. Similarly, the need for accurate and timely information collection and analysis are essential for the health plan management to determine eligibility and to assess patterns of care being delivered.

Measured against the needs of both the providers and the administrators, the classical paper-based medical record is often grossly deficient, cumbersome, and expensive to maintain and control. The over-all objectives of the project are to develop a computer-based system that can serve both primary patient care and administrative needs, be less expensive both in terms of provider time and medical record room costs, and provide a base for expansion to eventually support all the information processing needs of the HCHP.

The particular advantages a computer-based system offers include the following:

1. An accurate, up-to-date, and readily accessible registration and patient identification system is maintained. The membership file is the source of administrative data which provides health center management with statistical and billing information for successful management and planning.
2. The medical record information is not stored in a single physical document which is available at only one geographical location at any single point in time; instead, the information is stored in a dynamic information base which can be instantaneously updated and displayed from many different locations simultaneously.
3. The computer system can be used to organize the medical information by any of a number of different rules or algorithms. The information can be summarized to correspond to the particular problems of the patient or the particular needs of a specialty in sharp contrast to the strict chronological organization of the classical paper-based system.
4. The computer system assists in the primary care process by collecting existing data which should be brought to the provider's attention, e.g., reports of all abnormal laboratory test results, identification of patients receiving certain medications which should be discontinued, and identification of patients with certain characteristics warranting special consideration.
5. The computer-based system facilitates the HCHP commitment to an active program of quality assurance by monitoring particular case activities as to conformance to pre-defined standards of patient care. When deviations from the specified standards occur, the computer system can flag this information, and bring it to the attention of the provider.

6. The importance of supporting services such as laboratory test reporting, patient scheduling activity, and claims processing is recognized in this unified system since data need be collected and entered only once to be available for all functions.
7. The availability of a comprehensive, accessible data base is an essential factor to facilitating research into the delivery problems of ambulatory care, the promise of which was one of the essential motivating factors which originally led to the establishment of the HCHP.
8. The management, filing, assembling, and distribution of paper medical records can be a costly procedure, since it is labor-intensive, and since above a certain level even marginal improvements reflect significant cost increments. The expectation is that the computer-based system will prove to be a less expensive method of collecting, storing and retrieving this information.

In summary, the computer-based medical record system developed at HCHP satisfies the dual goals of providing both the data necessary to keeping valid, timely, and readily accessible information needed for routine and emergency patient care in an ambulatory practice, and the data necessary for management and supervision of a health care delivery system. It must be emphasized, however, that the computer-based system is a developmental activity. The essence of the system is not just its technological challenge, but its radically different approach to the combination of data recording and retrieval patterns by its users.

EVOLUTION

It should be understood that the HCHP M/MIS is not the result of a recent one-time project. Rather it is an evolutionary development effort that is dedicated to a continuing improvement in health care services through the use of computerized medical records.

This record system has evolved in phases, with the latest enhancements reflecting the fourth phase. Each step has allowed the realization of additional record system objectives while maintaining established system reliability and while simultaneously, the feasibility of the next evolutionary step was being explored.

The first phase, which began in 1969, was in essence a duplicate system using both a paper-based medical record and a computer-based information system. Originally, this redundancy was required because the reliability and provider acceptance of a computer-based system had yet to be demonstrated.

Initially only data entry was on-line, being executed from Teletype terminals in the medical records room. All retrieval of computer-based records was in batch mode. Although it obviously had far to go to be ideal, the first phase did allow the achievement of several record system objectives. A structured, organized, legible medical record was instituted as the basis for peer review activities. In addition, it was firmly established that a computerized medical record was acceptable to physicians and nurses at the HCHP.

Significant difficulties, however, were encountered in maintaining the duplicate system. Whereas data collection procedures for the computer system worked well, the paper-based system never kept up-to-date the assembling and filing of the profusion of patient transactions. Neither could the paper record be delivered in time for non-scheduled visits or telephone encounters. The paper record was often incomplete, particularly regarding care activities in the recent year. When complete, it was too frequently unavailable to the provider. In addition, it became obvious that the manual collating, filing, and assembling of information necessary to the maintenance of an up-to-date record was one of the most expensive activities of the record system.

In 1972, a second phase of development introduced Cathode Ray Tubes into the clinical areas for the first time, thus allowing a physician or nurse to directly recall a summary subset of a patient's medical record from the computer's files.[1] In 1973 a phase was introduced allowing providers to search through all encounter reports (a record of the patient's visit) and to flow chart (chronologically list) diagnoses, test results, and vital signs. This offered providers the opportunity of directly accessing a patient's complete computer medical record when the paper record was unavailable or incomplete.

A vital benefit of phases one through three was to prove that it was possible to realize a high degree of real-time system reliability and that the HCHP providers could work successfully with a computer-based system. With this encouragement, a fourth phase was begun—a system designed to maximize the use of computer-stored information and to minimize the use of the paper-based document. The most significant enhancement was the approval to greatly expand the medical content of the computer-based information system by including the physician's dictated comments and use the computer data bank as the primary storage repository for medical records information.

MEDICAL SYSTEM

The fourth and latest phase of development will be implemented in February, 1974. Basically this phase includes the addition of dictation provisions and new expanded encounter forms. The primary objective of the new medical system is that the relevant medical information be available to the provider at or before patient arrival (encounter).

This information will be made available to the provider in two different formats: (a) as computer-generated reports when there is several hours notice as in the case of scheduled visit, or (b) on the CRT for all telephone consultations, walk-in visits or when additional information is desired to supplement computer-generated reports.

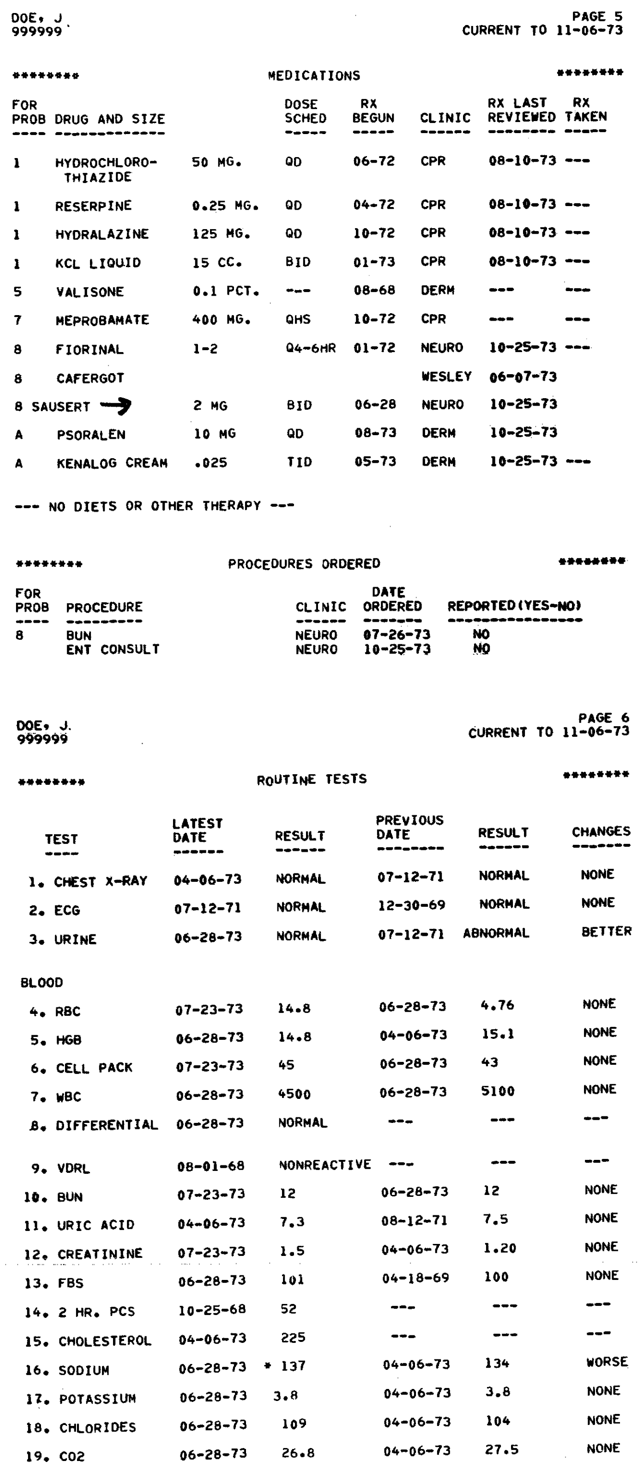
Patients cause an encounter report to be added to the medical record by a visit or a telephone consultation with a provider or other health center professional. In each case,

```
DOE, J                                                    PAGE 1
999999                                          CURRENT TO 11-06-73
NEXT VISIT TO        11-10-73                    PRINTED 11/20/73


                           NUCRSS-5

                    CLINICAL RECORD SUMMARY
                           NUMBER 5




   PLEASE LOOK AT THE LAST PAGE AND CONSIDER THE COMMENTS
   AND SUGGESTIONS.  FEEL FREE TO CORRECT OR COMMENT ON THIS SUMMARY.

                                     THANK YOU,

                                     DR. OLGA M. HARING




                                         TABLE OF CONTENTS
                                         -----------------

                                    PROBLEM LIST:        PAGE 3

                                    VITAL SIGNS:         PAGE 4

                                    CARDIAC-PULMONARY-
                                      RENAL DIAGNOSES:   PAGE 4

                                    TREATMENT:           PAGE 5
UPDATED:
      -----                         ROUTINE LAB TESTS:   PAGE 6
REVIEWED:
      -----                         OTHER LAB TESTS:     PAGE 7
KEYPUNCHED:
        -----                       SUGGESTIONS:         PAGE 8


                                                          PAGE 2
DOE, J.                                         CURRENT TO 11-06-73
999999

********             PATIENT IDENTIFICATION               ********



                                        SOC. SEC. NO.  000-10-2000
JOHN DOE,                               MALE           69 IN.
 37 YR. OLD      BLACK                  PHONE:         XX0-0000
0000 NOWHERE, U.S.A.


FIRST VISIT TO NUMC    07-30-68   -PRECLIN
LAST VISIT TO NUMC     08-02-73   -  DERM


LAST COMPLETE P.E.       04-13-73


DRUG ALLERGIES AND
  IDIOSYNCRASIES




********             LAST HOSPITAL ADMISSION              ********


HOSPITAL             PMH

DATE ADMITTED        12-29-68          DATE DISCHARGED      01-10-69

DISCHARGE DIAGNOSES

        01. GANGLION LT. WRIST AND LT. FOOT
        02. ESSENTIAL HYPERTENSION
```

Figure 1

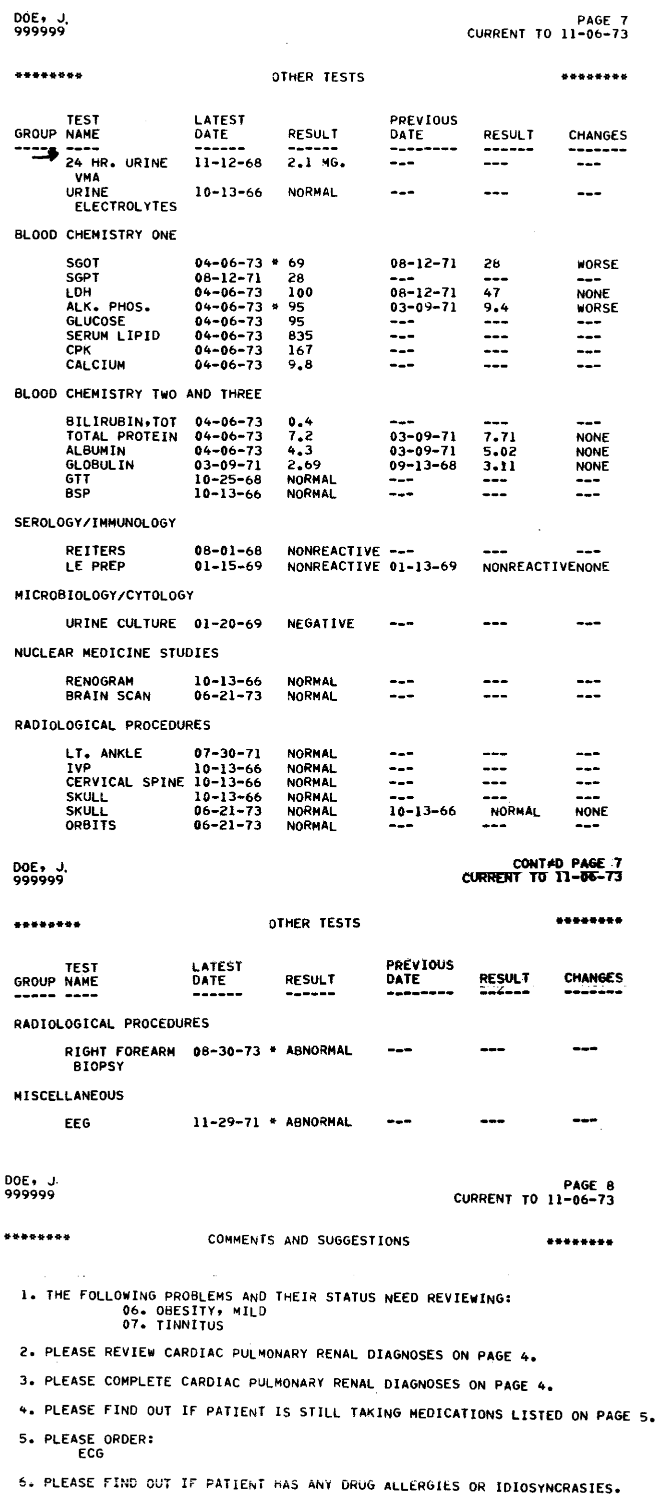
```
DOE, J                                                         PAGE 3
999999                                               CURRENT TO 11-06-73

********           PROBLEMS IN ALL ORGAN SYSTEMS              ********
                                 AND
                      VISITS TO MANAGING CLINICS

                        DATE      MANAGING  LAST
  PROBLEM               NOTED     CLINIC    NOTED     STATUS    DISPOSITION
  -------               -----     --------  -----     ------    -----------
01. HYPERTENSION        07-30-68  CPR       07-12-73  ACTIVE    RX ON P 5

02. ALCOHOLISM          05-07-69  CPR       05-21-71  ---       ---

03. LEUKODERMA          07-30-68  CPR       08-02-73  ACTIVE    RX ON P 5

04. SPRAIN,             08-30-71  ORTHO     ---       RESOLVED  ---
     LT. ANKLE

05. DYSHIDROSIS,        08-13-68  DERM      08-13-68  ---       RX ON P 5
     LT. HAND

06. OBESITY, MILD       09-09-68  CPR       09-09-68  ACTIVE    ---

07. TINNITUS            10-13-72  CPR       10-13-72  ACTIVE    RX ON P 5

08. HEADACHES           09-10-71  NEURO     10-25-73  ---       RX ON P 5



TEMPORARY PROBLEMS


 A. INFLAMMATORY        04-19-73  DERM      08-02-73  ACTIVE    RX ON P 5
     SKIN CHANGES


DOE, J.                                                        PAGE 4
999999                                               CURRENT TO 11-06-73


********               VITAL SIGNS AT RECENT VISITS            ********

  DATE    CLINIC   DOCTOR+STUDENT  WT   BP-SUPINE     PULSE-RAD  RESP  TEMP
  ----    ------   --------------  --   ---------     ---------  ----  ----

04-13-73  CPR      BROWN           174  130/92        ---        ---   98.4

01-19-73  CPR      LEVINE          176  140/110       ---        ---   98.8

10-13-72  CPR      WALKER          177  148/114 SIT   76         16    98.4

08-11-73  CPR      SMITH           175  140/118 SIT   ---        ---   98.6

07-14-72  CPR      BROWN           173  130/90        92         16    98.6

06-16-72  CPR      BROWN           175  150/104 SIT   68         18    98.0

06-09-72  CPR      BROWN           174  150/98        60         16    97.8

04-14-72  CPR      BROWN           179  160/120       72         16    98.0

01-21-72  CPR      HUNTER          175  150/108       72         16    98.6


********             CARDIAC-PULMONARY-RENAL CLINIC             ********

FIRST VISIT TO CPR        10-25-68
LAST VISIT TO CPR         04-13-73         CPR VISIT SCHEDULED FOR  11-10-73
  DOCTOR:                   BROWN
  STUDENT:                  ---

                                 DIAGNOSES
                                 ---------

HEART

   HEART             POTENTIAL HEART DISEASE

   ETIOLOGY          HYPERTENSION

   ANATOMY

   PHYSIOLOGY

   FUNCTIONAL CLASSIFICATION
   THERAPEUTIC CLASSIFICATION
   CLASSIFICATION REVIEWED

CIRCULATION          HYPERTENSION, ESSENTIAL

LUNGS                NORMAL

KIDNEYS              NORMAL
```

Figure 1 (continued)



```
DOE, J                                                    PAGE 5
999999                                         CURRENT TO 11-06-73

********                    MEDICATIONS                   ********

FOR                                DOSE    RX             RX LAST  RX
PROB DRUG AND SIZE                 SCHED   BEGUN  CLINIC  REVIEWED TAKEN
---- -------------                 -----   -----  ------  -------- -----

1    HYDROCHLORO-     50 MG.       QD      06-72  CPR     08-10-73 ---
      THIAZIDE

1    RESERPINE        0.25 MG.     QD      04-72  CPR     08-10-73 ---

1    HYDRALAZINE      125 MG.      QD      10-72  CPR     08-10-73 ---

1    KCL LIQUID       15 CC.       BID     01-73  CPR     08-10-73 ---

5    VALISONE         0.1 PCT.     ---     08-68  DERM    ---      ---

7    MEPROBAMATE      400 MG.      QHS     10-72  CPR     ---      ---

8    FIORINAL         1-2          Q4-6HR  01-72  NEURO   10-25-73 ---

8    CAFERGOT                                     WESLEY  06-07-73

8 SAUSERT →           2 MG         BID     06-28  NEURO   10-25-73

A    PSORALEN         10 MG        QD      08-73  DERM    10-25-73

A    KENALOG CREAM    .025         TID     05-73  DERM    10-25-73 ---

--- NO DIETS OR OTHER THERAPY ---


********                 PROCEDURES ORDERED                ********

FOR                                        DATE
PROB   PROCEDURE                   CLINIC  ORDERED   REPORTED(YES-NO)
----   ---------                   ------  -------   ----------------
8      BUN                         NEURO   07-26-73    NO
       ENT CONSULT                 NEURO   10-25-73    NO


DOE, J.                                                    PAGE 6
999999                                          CURRENT TO 11-06-73

********                  ROUTINE TESTS                    ********

                   LATEST                 PREVIOUS
  TEST             DATE       RESULT      DATE       RESULT    CHANGES
  ----             ------     ------      --------   ------    -------

 1. CHEST X-RAY    04-06-73   NORMAL      07-12-71   NORMAL     NONE

 2. ECG            07-12-71   NORMAL      12-30-69   NORMAL     NONE

 3. URINE          06-28-73   NORMAL      07-12-71  ABNORMAL    BETTER

BLOOD

 4. RBC            07-23-73   14.8        06-28-73   4.76       NONE

 5. HGB            06-28-73   14.8        04-06-73   15.1       NONE

 6. CELL PACK      07-23-73   45          06-28-73   43         NONE

 7. WBC            06-28-73   4500        06-28-73   5100       NONE

 8. DIFFERENTIAL   06-28-73   NORMAL        ---       ---       ---

 9. VDRL           08-01-68   NONREACTIVE ---         ---       ---

10. BUN            07-23-73   12          06-28-73   12         NONE

11. URIC ACID      04-06-73   7.3         08-12-71   7.5        NONE

12. CREATININE     07-23-73   1.5         04-06-73   1.20       NONE

13. FBS            06-28-73   101         04-18-69   100        NONE

14. 2 HR. PCS      10-25-68   52            ---       ---       ---

15. CHOLESTEROL    04-06-73   225           ---       ---       ---

16. SODIUM         06-28-73 * 137         04-06-73   134        WORSE

17. POTASSIUM      06-28-73  3.8          04-06-73   3.8        NONE

18. CHLORIDES      06-28-73   109         04-06-73   104        NONE

19. CO2            06-28-73   26.8        04-06-73   27.5       NONE
```

Figure 1 (continued)

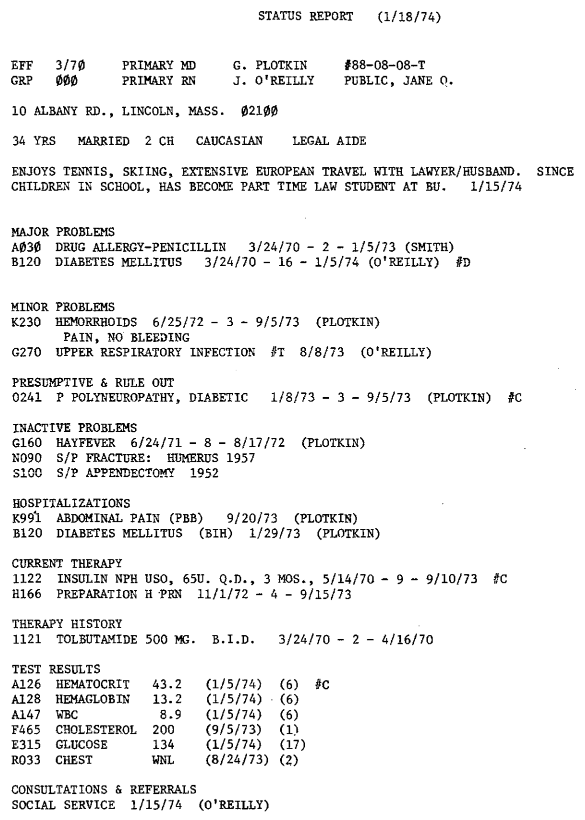


```
DOE, J.                                                    PAGE 7
999999                                          CURRENT TO 11-06-73

********                   OTHER TESTS                     ********

      TEST           LATEST                   PREVIOUS
GROUP NAME           DATE       RESULT        DATE       RESULT    CHANGES
----→ ----           ------     ------        --------   ------    -------
      24 HR. URINE   11-12-68   2.1 MG.       ---        ---       ---
       VMA
      URINE          10-13-66   NORMAL        ---        ---       ---
       ELECTROLYTES

BLOOD CHEMISTRY ONE

      SGOT           04-06-73 * 69            08-12-71   28        WORSE
      SGPT           08-12-71   28            ---        ---       ---
      LDH            04-06-73   100           08-12-71   47        NONE
      ALK. PHOS.     04-06-73 * 95            03-09-71   9.4       WORSE
      GLUCOSE        04-06-73   95            ---        ---       ---
      SERUM LIPID    04-06-73   835           ---        ---       ---
      CPK            04-06-73   167           ---        ---       ---
      CALCIUM        04-06-73   9.8           ---        ---       ---

BLOOD CHEMISTRY TWO AND THREE

      BILIRUBIN,TOT  04-06-73   0.4           ---        ---       ---
      TOTAL PROTEIN  04-06-73   7.2           03-09-71   7.71      NONE
      ALBUMIN        04-06-73   4.3           03-09-71   5.02      NONE
      GLOBULIN       03-09-71   2.69          09-13-68   3.11      NONE
      GTT            10-25-68   NORMAL        ---        ---       ---
      BSP            10-13-66   NORMAL        ---        ---       ---

SEROLOGY/IMMUNOLOGY

      REITERS        08-01-68   NONREACTIVE ---          ---       ---
      LE PREP        01-15-69   NONREACTIVE 01-13-69     NONREACTIVENONE

MICROBIOLOGY/CYTOLOGY

      URINE CULTURE  01-20-69   NEGATIVE      ---        ---       ---

NUCLEAR MEDICINE STUDIES

      RENOGRAM       10-13-66   NORMAL        ---        ---       ---
      BRAIN SCAN     06-21-73   NORMAL        ---        ---       ---

RADIOLOGICAL PROCEDURES

      LT. ANKLE      07-30-71   NORMAL        ---        ---       ---
      IVP            10-13-66   NORMAL        ---        ---       ---
      CERVICAL SPINE 10-13-66   NORMAL        ---        ---       ---
      SKULL          10-13-66   NORMAL        ---        ---       ---
      SKULL          06-21-73   NORMAL        10-13-66   NORMAL    NONE
      ORBITS         06-21-73   NORMAL        ---        ---       ---

DOE, J.                                             CONT#D PAGE 7
999999                                          CURRENT TO 11-06-73

********                   OTHER TESTS                     ********

      TEST           LATEST                   PREVIOUS
GROUP NAME           DATE       RESULT        DATE       RESULT    CHANGES
----- ----           ------     ------        --------   ------    -------

RADIOLOGICAL PROCEDURES

      RIGHT FOREARM  08-30-73 * ABNORMAL      ---        ---       ---
       BIOPSY

MISCELLANEOUS

      EEG            11-29-71 * ABNORMAL      ---        ---       ---


DOE, J.                                                    PAGE 8
999999                                          CURRENT TO 11-06-73

********             COMMENTS AND SUGGESTIONS              ********


 1. THE FOLLOWING PROBLEMS AND THEIR STATUS NEED REVIEWING:
             06. OBESITY, MILD
             07. TINNITUS

 2. PLEASE REVIEW CARDIAC PULMONARY RENAL DIAGNOSES ON PAGE 4.

 3. PLEASE COMPLETE CARDIAC PULMONARY RENAL DIAGNOSES ON PAGE 4.

 4. PLEASE FIND OUT IF PATIENT IS STILL TAKING MEDICATIONS LISTED ON PAGE 5.

 5. PLEASE ORDER:
       ECG

 6. PLEASE FIND OUT IF PATIENT HAS ANY DRUG ALLERGIES OR IDIOSYNCRASIES.
```

Figure 1 (continued)

new information is captured by the provider on an encounter form which has now been expanded to allow more self-encoding and to include biographic information. Each encounter is maintained as a separate entry in the medical record. All data, including dictation, from any encounter is available upon request either on hard copy or as a CRT display.

Information such as laboratory test results, X-ray findings,



```
                              STATUS REPORT   (1/18/74)

EFF   3/7Ø      PRIMARY MD     G. PLOTKIN     #88-08-08-T
GRP   ØØØ       PRIMARY RN     J. O'REILLY    PUBLIC, JANE Q.

10 ALBANY RD., LINCOLN, MASS.  Ø21ØØ

34 YRS   MARRIED  2 CH   CAUCASIAN    LEGAL AIDE

ENJOYS TENNIS, SKIING, EXTENSIVE EUROPEAN TRAVEL WITH LAWYER/HUSBAND.  SINCE
CHILDREN IN SCHOOL, HAS BECOME PART TIME LAW STUDENT AT BU.   1/15/74


MAJOR PROBLEMS
AØ3Ø  DRUG ALLERGY-PENICILLIN   3/24/70 - 2 - 1/5/73 (SMITH)
B120  DIABETES MELLITUS   3/24/70 - 16 - 1/5/74 (O'REILLY)  #D


MINOR PROBLEMS
K230  HEMORRHOIDS  6/25/72 - 3 - 9/5/73  (PLOTKIN)
       PAIN, NO BLEEDING
G270  UPPER RESPIRATORY INFECTION  #T  8/8/73  (O'REILLY)

PRESUMPTIVE & RULE OUT
0241  P POLYNEUROPATHY, DIABETIC   1/8/73 - 3 - 9/5/73  (PLOTKIN)  #C

INACTIVE PROBLEMS
G160  HAYFEVER  6/24/71 - 8 - 8/17/72  (PLOTKIN)
N090  S/P FRACTURE:  HUMERUS 1957
S100  S/P APPENDECTOMY  1952

HOSPITALIZATIONS
K991  ABDOMINAL PAIN (PBB)   9/20/73  (PLOTKIN)
B120  DIABETES MELLITUS  (BIH)  1/29/73  (PLOTKIN)

CURRENT THERAPY
1122  INSULIN NPH USO, 65U. Q.D., 3 MOS., 5/14/70 - 9 - 9/10/73  #C
H166  PREPARATION H PRN  11/1/72 - 4 - 9/15/73

THERAPY HISTORY
1121  TOLBUTAMIDE 500 MG.  B.I.D.   3/24/70 - 2 - 4/16/70

TEST RESULTS
A126  HEMATOCRIT   43.2   (1/5/74)  (6)  #C
A128  HEMAGLOBIN   13.2   (1/5/74)  (6)
A147  WBC           8.9   (1/5/74)  (6)
F465  CHOLESTEROL  200    (9/5/73)  (1)
E315  GLUCOSE      134    (1/5/74)  (17)
R033  CHEST        WNL    (8/24/73) (2)

CONSULTATIONS & REFERRALS
SOCIAL SERVICE  1/15/74  (O'REILLY)
```

Figure 1 (continued)

and ECG results are, of course, recorded on separate forms designed to code the appropriate results.

The status report (Figure 1) is a summary of the patient's medical history and can be viewed as both a brief summary of important administrative and medical data which can be scanned in seconds and as a table of contents to the patient's complete medical record. Nine categories of information are contained in this important document:

1. Registration information
2. Primary provider
3. Social and demographic data
4. Diagnoses and problems list
5. Hospitalizations
6. Current therapy
7. Therapy history
8. Test results
9. Consultations and referrals.

This group of computer-generated reports provided before a patient visit will include all information deemed relevant by each specialty group. For example, in internal medicine, it will be comprised of the status report, the last encounter report to the primary provider, all intervening encounter reports, the last encounter report for each major problem, and a laboratory test summary. The service-level design goal is that for over 90 percent of the typical patient care visits, the computer reports generated and delivered prior to an encounter will be sufficient. If further information is needed, the CRT may be used to access all previously recorded medical information stored in the computer.

As a better understanding of the particular needs of each specialty group and of particular medical problems is reached through the use of the system and feedback from its users, other standard computer-generated reports will be created. For example, flowchart presentations—the chronological listing of the diagnoses, tests, or therapies related to a specific problem of a specific patient—will be provided routinely for pediatric visits (e.g., height and weight growth charts) and for prenatal visits. In process of development is a flowchart presentation for the routine management of the patient with hypertension.

No medical records system is adequate that has not provided for prompt updating of patient records. To be effective and establish credibility with the user, the record must be current and comprehensive in addition to being accurate. A control system has been developed by the HCHP management that ensures that an encounter form has been input within established time limits.

MANAGEMENT SYSTEM

In addition to medical information the patient file contains administrative data which provides health center management with statistical and billing information vital to successful management and planning. Enrollment in the Plan creates the demographic and registration portion of the patient's file. This nucleus, combined with the information entered from each medical encounter, establishes the basis for the management information system.

A variety of administrative and management functions are necessary and available. Mailing labels are generated to send medical history questionnaires and health center information to subscribers. Forms are produced which are used for the generation of membership identification cards. Membership information becomes the control system for capitation billing as well as fee-for-service billing of non-members.

Hospitalizations and outside referrals are noted by the provider on the encounter form, input to the computer, and made available on inquiry to verify patient insurance claims.

Using the population as the denominator, utilization and membership statistical reports are available. For example, certain reports can provide data organized by age, sex, geographic location, medical specialties involved or any combination thereof.

Cost/Membership ratios allow management emphasis on specific cost centers. Facility utilization can be assessed via patient encounter statistics. Management, thus informed, can better plan for future requirements including "what-if" projections related to potential numbers of future memberships.

Special studies allow personalized attention. A simple example is the production of mailings notifying members satisfying certain medical criteria of their need for flu shots.

These are some of the features which provide the HCHP management with the information services necessary to efficient, effective operations, controls, and planning.

QUALITY ASSURANCE—A FUTURE OBJECTIVE

The primary objective of the medical record system described in considerable detail in the preceding pages is to ensure that readily accessible, legible and accurate information is available for each and every patient seeking medical services at the Harvard Community Health Plan. An important by-product of this endeavor will be the potential of the system as a tool for monitoring the general quality of the services that are delivered.

As you are aware, the profession has entered an era in which it will not be enough to practice good medical care—it is now expected to *demonstrate* in rather precise ways that this is the case. The federal authorities have already enacted legislation designed to ensure that the services which they fund, such as Medicare and Medicaid, are of acceptable quality. It is therefore safe to predict that in the near future Blue Cross/Blue Shield and commercial insurance carriers will do likewise. Moreover, consumer groups, which are becoming organized at the Harvard Community Health Plan, will be making similar demands.

It is anticipated that, when properly used, reliable and comprehensive data available in the computer will enable almost complete obviation of the laborious process of individual manual medical record review when appraising the quality of care in various clinical situations.

Furthermore, numerous aspects of utilization, such as prescribing patterns, the keeping of appointments, over and under-utilization, membership turnover, the use of hospital facilities and many other matters will be available in a form more accurate, more comprehensive, and more accessible than was previously the case.

It seems probable that in the near future, the whole process of quality assurance, including peer review and utilization, will, with the help of the new record system, become part of the ongoing information system at the Harvard Community Health Plan.

TECHNICAL CONSIDERATIONS

A universal characteristic of information systems is that objectives and procedures will change with time. In part these changes are related to inadequacies in the initial planning stages. In a large sense, however, they are related to the inherent effect of a successful introduction of computer technology into a new field. It is the hallmark of such a system that the users modify their initial attitudes, raise their expectations and make increasing demands on information handling services. This general characteristic, and others unique to medical records systems, should prompt the systems designer to implement in a high level language.[2] Certain specific characteristics of this language system should also be considered:

1. It should be procedural, one which can be easily modularized.
2. It should have powerful string processing capabilities. Much of the information in a medical system is textual in nature; names, clinical results, free text dictation.
3. It must be able to search and manipulate data quickly and easily. I/O should be flexible, allowing for experimentation in formatting and terminal selection.
4. It must have the capability for the development and maintenance of a large data base. A data management system for patient care requires a relatively complex data base for several reasons:
   (a) The data can assume a variety of types and formats.
   (b) The data items are dynamic in size; fixed dimensioning of data or data fields is unacceptable.
   (c) The file must be organized for rapid easy access of specific sections, e.g., one is often interested in a particular encounter or laboratory value rather than in a whole patient file.

The designers of the Harvard M/MIS feel that the MUMPS system meets or exceeds all these requirements. MUMPS (Massachusetts General Hospital Utility Multiprogramming System)[3] is a compact time-sharing system implemented at the Laboratory of Computer Science on Digital Equipment Corporation PDP-9's and PDP-15's. The interpreted MUMPS language includes extensive capabilities for string processing, terminal I/O and manipulation of a dynamic, hierarchical data base. For the project described here, if one also includes criteria of cost and availability, there is effectively no competition.

CONCLUSION

The joint Laboratory of Computer Science/Harvard Community Health Plan Medical/Management Information System venture represents a successful advancement in the use of one technology to enhance the effectiveness of another. Project experience indicates that, properly served, the goals of administration and the medical profession are not divergent but concurrently support quality health care. Careful consideration of the users needs succeeded in dispelling skepticism and has paid magnificent dividends in the user satisfaction realized since implementation.

Finally, credit must be given to a strong HCHP management commitment to the development of M/MIS. It took this commitment and a generous ration of faith to overcome the obstacles, setbacks and frustrations of this long development and hold firmly to the goals they envisioned. The investment has been high, but it is our conviction that both the system presently in existence and the potential of the system yet to be realized justify this development. All of us envision the day when the paper medical record will be only a limited archival storage and an instantly retrievable computer medical record will be the operational mainstay of the medical profession.

REFERENCES

1. Grossman, J. H., G. O. Barnett, T. D. Koepsell, H. R. Nesson, J. L. Dorsey and R. R. Phillips, "An Automated Medical Record System," *J.A.M.A.*, Vol. 224, No. 12, pp. 1616-1621, 1973.
2. Barnett, G. O., "The Modular Hospital Information System," *Computers in Biomedical Research*, Fourth Volume, edited by Bruce Waxman, Ph.D. and Ralph W. Stacey, Academic Press, in press.
3. Greenes, R. A., A. N. Pappalardo, C. W. Marble and G. O. Barnett, "Design and Implementation of a Clinical Data Management System," *Computers in Biomedical Res.*, 2, pp. 469-485, 1969.

A status report on the TICCIT project

by C. VICTOR BUNDERSON

Brigham Young University
Provo, Utah

In September of this year, two TICCIT systems installed at Phoenix College and the Alexandria Campus of Northern Virginia Community College will be operating with students in mathematics and English courses. This will begin the first phase of a two-year study involving the evaluation, improvement, and the demonstration of the concepts of learner-controlled courseware administered by a low-cost computer-controlled television system.

The TICCIT system, unlike other CAI systems produced by manufacturers or engineering-oriented laboratories, is designed around a set of educational goals and instructional principles. Goals for institutions, including low cost, reduced time to complete material, and increased enrollment, influenced the design of TICCIT. The goals also include content goals related to the mathematics and English content programmed for the system. Perhaps the most innovative goals are those for students, which include mastery, efficiency, improved learning strategies, improved attitudes of approach rather than avoidance, and responsibility. Other goals relate to creation of some new roles of teachers and other educators who will be involved with this complex system. For computer-assisted instruction to survive in existing educational institutions, it must serve the needs of teachers, as well as the needs of students.

Developmental versions of the TICCIT system have been in operation at Brigham Young University for more than a year. During the last few months, software became available so that editing, debugging, and student tests could be accomplished on the system. Preliminary generalizations from the authoring process, student tests and from other experiences using this evolving system are the subjects of this paper.

The presentation was organized around the goals described above. First of all, the hardware and software were briefly described and their relationship to the institutional goals were delineated. Second, experiences with the system as an innovation in courseware authoring, inputting, debugging, and evaluating student data were discussed. While it was still too early to discuss data from student tests, which had barely begun as this conference convened, an informal description of students' early reaction to a learner control command language was given.

One result of the TICCIT project has been to develop an implementation plan describing, among other things, the proposed new roles of teachers in the new system. Progress toward the definition of teachers' roles in this system was briefly described.

What classroom role should the PLATO computer system play?

by ROBERT B. DAVIS

University of Illinois
Urbana, Illinois

Inserting computers into the ecology of an elementary school classroom involves a combination of promise and uncertainty that parallels similar technological innovations in other areas, whether heart pacers, artificial kidneys, tranquilizers, atomic power plants, transportation, food production or virtually any other area one can think of. In each instance we lack a complete description of the original ecology, and we cannot be, *a priori*, fully aware of new possibilities.

This note explores a small part of this territory in the case of the PLATO computer system, as used in relation to elementary school mathematics and reading. Our purpose is to emphasize the large range of possible roles that computers might play, probably with varying degrees of effectiveness. Because of the many possibilities it should become clear that the question "Can PLATO teach?" is improperly phrased, and should be replaced by the question "What useful roles can PLATO play in the classroom?"

THE PLATO SYSTEM

From a student's point of view, the PLATO computer system is a terminal with a screen somewhat like a television set, plus a keyboard somewhat like a typewriter. In fact, the screen is a plasma panel, consisting of a quarter of a million tiny, independently controlled points of light, in the form of minute bubbles of neon gas. Because the plasma panel uses digital data, it is absolutely free from distortion of the kind that limits the usefulness of CRT's. The information rate into each terminal precludes showing "movies" a la TV, but a considerable amount of animation is possible—a train can run across the screen, for example, or a bird could fly across. A touch panel allows the computer to known where a child touches the panel, if he does. A random access audio unit allows the computer to "talk" to students with good quality reproduction of the human voice, or of other sounds. Slides can be shown on the screen, via rear-view projection.

In fact, although students are ordinarily unaware of it, these terminals are connected to a time-shared computer (CDC 6000 series). The screen can be thought of as the page of a book on which the computer can write or draw, the child can write or draw, and modest animation is possible, plus pictures from the rear-view projection of slides. The use of a large time-shared computer means that many programs or "lessons" are simultaneously available (the "Library of Congress" effect), together with a considerable amount of computing power. The computer keeps records on the performance of each student during his or her previous sessions with the terminal.

THE "NATIONAL DEMONSTRATION" OF PLATO

Obviously, the PLATO computer system is a powerful and flexible tool that should be capable of playing many useful roles in the classroom. To begin to test these possibilities, an official "demonstration" is under way, costing approximately 8 million dollars. During the academic years September 1974—June 1975 and September 1975—June 1976 PLATO will be in operation with a reading program in kindergarten and grade one, and a mathematics program in grades 4, 5, and 6, with 100 terminals in elementary schools in Champaign and Urbana, Illinois. The results will be observed and described by Educational Testing Service, of Princeton, New Jersey.

The goal of this demonstration is to identify one or more useful roles that PLATO can play in elementary school classrooms.

CHILDREN'S MATHEMATICAL THOUGHT

As one part of the job of getting ready for this demonstration, the PLATO courseware group has been studying the mathematical thinking of children who are in grades 5, 6, and 7 in an individualized paper-and-pencil school mathematics program that does NOT use computers.[1] This has given us a direct view of how these children solve and discuss various mathematical problems, from which we can infer a great deal about the way they think about mathematics. From this, and from direct observation of the school program, we can infer what transactions are taking place in the classroom, and how the cumulative experience with these transactions is effecting the child's thinking.

One typical result is the following: the school program presents the child with a pamphlet showing an illustrative example, followed by problems of the same type for the child to work out. When completed, this pamphlet is turned in, corrected, and returned to the student with an indication of which problems were solved correctly, and which are wrong. Among the transactions that do *not* occur are: students talking about their work to adults or to other students, experience in physical uses of mathematics (as in making an accurate scale drawing of the school playground), and student diagnostic analysis to decide how to attack a problem (since problem sets are homogeneous).

A fifth-grade girl in the program, asked to add

$$.3 + .4 = \quad ,$$

wrote

$$.3 + .4 = .7 \ .$$

Asked to add

$$3. + 4. = \quad ,$$

she wrote

$$3. + 4. = 7. \ ;$$

but, asked to add

$$.3 + 4. = \quad ,$$

she wrote

$$.3 + 4. = .7. \quad .$$

How large is .7.? Is it bigger than 6, or smaller than 1? She did not know. The symbol .7. was of course meaningless to her, but this did not trouble her; many of the notations of mathematics were meaningless to her, and she had learned to accept that situation gracefully. To her, those little periods were something to be copied as one who does not speak French might copy accent marks in copying a French sentence.

This example is typical of students in this school program, and *in toto* such examples seem to support the assessment that math in this program was presented essentially as a matter of meaningless symbols, and learned as meaningless symbols.

THE ECOLOGY OF THE CLASSROOM

Assuming that what a child learns is influenced by the transactions in the classroom it becomes important to study these transactions. We are thus asking: "What can we learn about the ecology of the classroom BEFORE we allow computers to intrude there?"

As a suggestion of the variety of things that go on, we offer the following imcomplete list.

What do teachers do?

They arrange for a child to have a new experience (by taking the child to a zoo, or by showing a film, or by handing the child a thermometer, etc.)
They encourage the child to talk about that experience.
They orchestrate, if they do not compose, the curriculum (and some teachers compose it).
They explain something new by reference to things that are more familiar.
They arrange for student A to help student B.
They review things the child has done, and encourage him to reinterpret his experiences.
They demonstrate how to do something.
They suggest things to get a child started on a new line of thought.
They supervise.
They observe a student, and offer constructive criticism.
They show appreciation for student work or student discoveries.
They provide drill (as with flash cards for addition facts).
They lead a child to recognize some of the consequences of his thinking, a kind of elementary school adaptation of *reductio ad absurdam.*
They assign tasks.
They set goals.
They administer tests or other diagnostic procedures.
They help establish values and priorities.
They listen when a child needs an adult to talk to.
By their physical presence, they reassure.
They give a child the sense that someone cares about him, remembers who he is, recognizes him, and remembers what he did yesterday.
They set expectations.
They answer questions.
By their own behavior, they set an cxample (for instance, some children believe at first that in order to read you must know the story from memory beforehand, and they "read" this way, until it suddenly dawns on them that the adults *are doing something different*—namely, decoding the written symbols).
They guide a student performance.
They dole out helpful hints.
They influence the social reward system (but they cannot usually control it to the point of total denial of peer-group inputs).

But—a child is not alone in the room with the teacher. Other children play a major role. If, for example, a child hears other children recite, he may develop his ability to listen critically for weak spots in the other child's argument, or to identify hints that he can use himself. He will be subject to "social facilitation," acquiring goals from high-status children who have those goals. If he wishes to enter an argument, he may get practice in developing alternative conceptualizations, much as an attorney finds a way to construe a case. He is surely aware of how other children appraise his performance.

Perhaps the most important step a student takes is to accept a social contract to allow others to influence what he does and even how he thinks. The different extent to which two students do this may be the major difference between them.

Another important thing that students do is to work out their own rational explanation of what seems to be happening, in the form of explanatory rules or goals.

Students also, and to varying degrees, explore, discover, practice, make original creations, learn to take pride in their work, learn to organize their time, learn different ways to deal with different people or different situations, learn to resolve internal conflicts, and so on. Students may (not often enough) study problems in order to work out their own line of attack. They develop heuristic analysis strategies. They learn to have confidence in some things, and not to have confidence in others.

Typically, in mathematics, teachers do NOT explain a task clearly, but instead carry it out, and leave students the job of inferring the goal from observing and imitating the activity. A few teachers reverse this, and make sure the children have a clear understanding of the task, after which the teacher leaves it up to the child to devise a method of carrying out the task (this is one variant of "discovery" teaching).

Other things happen: teachers teach some things badly; some important items are left out altogether; teachers themselves learn a great deal in their own classrooms, about children in general, about specific individual children, and about the curriculum subjects (for example, from textbooks, reference books, etc.). Outside of the classroom, teachers learn from in-service courses and from independent study.

WHAT PLATO IS TRYING TO DO

Which of the items on our list—or on the much longer lists that can be made—should PLATO attempt to address? Which are logical, or "natural," tasks for PLATO? We can get some guidance from past PLATO experience: for example, it does *not* seem natural for PLATO to try to answer questions. Most questions are badly stated, and often very specific to that setting. Although PLATO offers CAI lessons on PLATO authoring in the TUTOR language, most novice programmers seek human question answerers, and their questions are often as specific as "Why is my program doing *this* (with a demonstration)?" Children's questions tend to be even more obscurely stated, and even more situation-specific. When one *really* has a question, usually part of the difficulty is that one is unable to state it clearly.

Can PLATO show a student the consequences of his own thinking, in a *reductio ad absurdam* fashion? We have one modest start in this direction, in a lesson on average velocity, authored by Bruce Sherwood, and designed for university students: if a student gives a wrong formula for average velocity—say, v_2-v_1—PLATO states a simple word problem ("A car accelerates uniformly from 40 mph to 60 mph. What is its average speed during this acceleration?"). Students nearly always answer this correctly (50 mph). Then PLATO responds: "But your formula v_2-v_1 gives 20 mph."

Can this technique be extended into elementary school? Possibly, but the misconceptions of elementary school children are considerably deeper and more elusive, and one would need a very clear presentation of the contradiction before one could convince them. (At present we are not attempting this.)

For the present 5th grade mathematics courseware, we are recognizing four aspects of mathematical knowledge, and explicitly pursuing three of them. The four are:

(i) knowing meanings (usually in concrete terms) of the various symbols, operations, concepts, etc.
(ii) skill in symbol manipulation
(iii) competence in using heuristic problem-analysis strategies
(iv) having appropriate attitudes and expectations.

We deal with the fourth, above, only implicitly; the first three we tackle explicitly.

Some of the methods used can be suggested by a few examples:

Example 1. "Darts and balloons." A vertical number line appears on the screen, with 0 and 1 indicated. The line can be interpreted as a wall, to which (using random numbers) PLATO attaches five balloons. If a student types ½ (or 0.5, or ¼ + ¼, and so on), a dart flies across the screen and thuds into the wall one-half of the way from 0 to 1. If it hits a balloon, the balloon bursts.

The simplest goal of this learning experience is to guarantee that children have a reasonable notion of the *size* of any common fraction. But more is possible; children transform this into many different lessons. One girl typed $\frac{1}{5}$, not near any balloon; but, with the distance from 0 to $\frac{1}{5}$ available to her as a unit, she measured this off with her fingers, and found $\frac{n}{5}$ for $n = 2, 3,$ and 4. If she found a balloon, she burst it. If any remained, she tried $\frac{1}{7}$; and so on.

One adult studied the tolerance—how close to the mid-line of a balloon must you hit in order to burst the balloon?

This should serve to remind us that learning experiences are complex things, not easily described, and not identifiable by brief statements of simple objectives.

Example 2. The Game "WEST." This is, in effect, a board game. The game board, and three spinners, appear on the PLATO screen. By pressing a key, the student "spins" the spinners, thus obtaining integer values for N_1, N_2, and N_3. Under simple rules (e.g., no operation sign used more than once), the player forms an expression (such as $N_1 \times (N_2 + N_3)$), and—provided he states the value of this expression correctly—he moves forward by this amount.

The evident explicit goal here is to provide a large amount of painless practice in arithmetic; but notice that there are also other goals: for example, to get students started thinking about the maximum value of such expressions.

Example 3. Names for Today's Date. In pursuing some of the possibilities for letting students create, letting them be pround of their work, letting them share and compete with one another, a lesson has been designed that says: "Today's date is November 7 (or whatever it is). What names can you make up for today's date?" The student now enters whatever names he chooses, such as

$$2 \times 3\tfrac{1}{2}$$

$$(\text{rnd}(\pi^2)) - (9)^{+1/2}$$

After he has entered as many names as he wishes, he presses the "NEXT" key, and PLATO displays the complete list of all names entered thusfar, including his own, with the names of the students who entered them. He now—having looked at the work of the other students—may enter still more names if he wishes.

Example 4. Programming PLATO. This lesson sequence also deals with allowing children to create within mathematics, much as they would in art or poetry—specifically, they can create original computer programs. The programming language is pictorial, and is developed by touching pictures on the PLATO screen. Sub-routines can be created, named, and used by name as instructions in future programs. A typical program might put trees in various locations, outline a street, then have a boy cross the street just in front of a car that drives down the street.

This is an experimental venture, the value of which may not be known for some time.

Example 5. The definition of fractions. What is interesting in this sequence is the underlying teaching strategy: the sequence begins with something children know very well indeed—whether a chocolate bar has been shared fairly, or not, among two, three, or four children. So the *action* is familiar; but while this familiar action is being carried on, it is being discussed in the language of fractions—thus, the first introduction of this language is by *use*, not definition. After some use, a new level is reached: operating on a "meta" level, PLATO and the student cycle back through what they have just done, but this time, instead of *doing* it, they *analyze* it.

Example 6. Interterminal Games. Two or more terminals can be interconnected (by the courseware) so that children can play games against live opponents, in real time.

Summary

Obviously, PLATO can attempt to address a fairly sizable range of typical classroom activities. Courseware is now being created to pursue a scattering of these. Presumably other classroom transactions would not be natural (or feasible) on PLATO. For the National Demonstration, we hope to show that there are some classroom tasks which we have correctly identified as appropriate and feasible via PLATO. On a few of our nominees we may fail to achieve success, either because the task is unsuitable, or else because we have not created appropriate courseware or appropriate conditions of use.

THE SESSION SELECTOR

On PLATO, one has great freedom in deciding whether choices are to be made by the children, or by the teacher, or by PLATO itself. When the choice is made by PLATO, the Session Selector program does it.

Good lessons, like good concerts and good chess games, have a beginning, a middle, and an end. The Session Selector plans PLATO lessons this way, choosing first the main course, which will appear to the student in the middle slot. This choice is the most carefully made, and utilizes the records of the individual student, plus the curriculum tree. After the choice for Slot II has been made, appropriate review or introductory material is chosen for Slot I; some appropriate games or other favorites of the children are then made available for Slot III. Although PLATO *plans* in the order II, I, III, the student of course encounters the slots in their usual order: the lesson begins with the Slot I selection, then goes on to Slot II, and ends with the Slot III games.

INTERACTIONS BETWEEN USE AND HARDWARE/SOFTWARE

It is probably obvious that the way PLATO is used in schools shapes the demands on the hardware and system software, and is in turn shaped by the capabilities of the hardware and the software. We cite one example: the allocations of extended core storage (ecs) on PLATO originally assumed that, on the average, twenty students were using the same lesson at the same time. A few years ago this might have been feasible—schools commonly had "a math period," "a reading period," and so on. It was not unusual for the teacher to say: "All right, children, now let's everyone turn to page 43 in our readers." Today, in the schools we are working with, this would be unusual. They have moved toward the "integrated day" approach; *time* is no longer subdivided, but *space* is—there is no "reading period," but there is a "reading corner," and there's always somebody over there reading. To accommodate to such schools, it has been necessary to rearrange memory allocation on PLATO, so that, on the average, it is necessary for four students to be using the same lesson. Since, when fully developed, PLATO may be serving 2,000 students at the same time, the 4-to-1 average may not impose too severe a restriction.

THE SHORT TERM VALUE OF PLATO

There are long-term hopes that PLATO may be an effective economy option, highly cost-effective because it makes the classroom more capital intensive and less labor intensive, and that the use of PLATO may significantly improve the *quality* of education. Both of these hopes may be realized, but probably not immediately. For the short term, PLATO is at a developmental stage, where the task confronting us is to see which classroom transactions we can handle well via PLATO, and to acquire the means of doing so.

But even in the short run, PLATO has considerable value as a research tool: as more classroom jobs are assigned to PLATO, we gain a far greater degree of control over what is going on in the classroom, which allows us to get far better data on the importance of different kinds of transactions. The will-o'-the-wisp elusiveness, subtlety, and complexity of traditional classrooms never allowed us, for example, to study the effect of omitting much or all of the usual arithmetic drill, inserting instead games such as WEST that provide less-controlled experience with arithmetical operations. In the traditional open classroom, as Featherstone[2] points out, one often found that every child had learned to read, but you could not identify when, where, or how this had taken place. PLATO should give us a far greater ability to pinpoint the contributions made by the various kinds of activities and transactions, which in turn allows one to plan the future role of PLATO on the basis of a more secure rational theory.

BIBLIOGRAPHY

1. Erlwanger, S. H., "Benny's Conception of Rules and Answers in IPI Mathematics," *Journal of Children's Mathematical Behavior*, Vol. 1, No. 2, Autumn, 1973, pp. 7-26.
2. Featherstone, J., *Schools Where Children Learn*, Liveright, NYC, 1971.
3. Davis, R., "Observing Children's Mathematical Behavior as a Foundation for Curriculum Planning," *Journal of Children's Mathematical Behavior*, Vol. 1, No. 1, winter, 1971-72, pp. 7-59.
4. Hammond, A. L., "Computer-Assisted Instruction: Two Major Demonstrations," *Science*, Vol. 176, June 9, 1972, pp. 1110-1112.
5. Bitzer, D. L., B. A. Sherwood and P. Tenczar, *Computer-Based Science Education*, CERL Report X-37, May, 1973, University of Illinois, Urbana, Illinois.

Computer assisted instruction comes of age in a public school system

by WILLIAM M. RICHARDSON

Montgomery County Public Schools
Rockville, Maryland

INTRODUCTION

During the late 1960's a number of public schools began experimenting with the development and use of computer-assisted and computer-managed instruction. Funding for these public school projects was provided primarily by Title III of the Elementary and Secondary Education Act of 1965, or other sources of federal funds. Due to the reduced availability of federal funding, few new public school CAI projects have been initiated since 1970. It is, however, very encouraging to analyze the results of the few active public schools CAI projects.

The paper will provide evidence that CAI/CMI can produce increased achievement when properly integrated into the instructional process. In deference to some of the early concerns with the application of computer technology, it will be shown that the computer can in fact provide greater individual and personalized instruction to students. Although computer-assisted and computer-managed instruction are not today cost-affordable when applied to all students within a school system, it will be shown how the use of CAI can be cost-justifiable for selected student target populations. In addition, public school systems have shown that they can effectively work with computer technology as both users and developers. The concluding premise of the paper is that the public education sector is rapidly approaching the time when it can effectively utilize widespread CAI as a direct aid to the instructional program.

BACKGROUND

The MCPS Title III CAI Project goals were the demonstration of the feasibility of computer-assisted instruction as an instructional medium and the assessment of its role in the K-12 public school setting. The project currently utilizes 31 time-shared computer terminals cable connected to the IBM 1500 instructional system. The project fulfilled its stated objectives by (1) developing, using, and evaluating over 40 modular instructional CAI and CMI packages, (2) providing the Montgomery County Public School System with a cadre of 70 individuals capable of developing and using CAI instruction, (3) providing an orientation to CAI to approximately 5,000 school administrators, supervisors, and teachers, and (4) making recommendations to MCPS concerning the future implementation of CAI.

Program design teams composed of project staff and classroom teachers developed approximately 40 modular instructional packages, most of which are in the mathematics and science areas. The instructional design for each program, including objectives, entering behaviors, hierarchy, and strategy has been completely documented in the *Project Reflect Title III Final Report*, June 30, 1972.

Federal funding for the project ended in June 1971. Since that date, MCPS has supported a staff of 12.5, maintenance on the computer and all additional expenses of the program. The manufacturer is providing the computer system lease free for instructional purposes.

RESULTS

Experiences over the six-year period have shown that learning and teaching philosophies could be altered, and that individualization with computer support is logistically feasible. When school faculties are provided with valid CAI/CMI materials related to student needs, teachers are able to integrate new technology into the regular instructional process. In this connection, experience has shown that teachers need time and training to effectively utilize these materials. In addition, it was found that selected teachers have the talents and interests to develop effective individualized CAI modules.

Year-long research studies on achievement, class size, and teacher-student interaction were completed in June 1972.

Results of these studies showed that:

1. Fifty-eight matched pairs of sixth-grade students with one-half hour weekly CAI experience as part of their regular arithmetic made significantly greater mean gains in achievement ($t = 2.08$, $df = 114$, $p < .05$);
2. High school students in two computer-managed geometry classes, which averaged over 33 students, showed no significant differences in mean gain scores than students in three traditional classes with an average of 23 students ($t = 1.23$, $df = 81$, n.s.); and

3. Secondary mathematics students enrolled in three classes with CAI and CMI support received significantly more individual attention from their teachers than students in three traditional classes ($F = 38.78$, $df = \frac{1}{4}$, $p < .01$).

In addition, a mini-study with a few special education students was conducted in one secondary school. Ten students from this school participated in a four-month study to determine if they could benefit from the use of the arithmetic materials prepared for the regular school population. Results of this study, in which the analysis of data was prepared as though the average I.Q. were normal, showed significantly greater gains in achievement than could be expected in the time period allotted ($t = 2.71$, $df = 9$, $p < .05$).

Evidence collected by the CAI Program and substantiated by other CAI installations showed that the hardware system and terminal components used for instruction must function in a reliable manner and that the response time for students must not exceed three (3) seconds.

IMPLICATIONS

It would appear that with minimal use at the elementary school level, students can be expected to increase their proficiency in basic arithmetic skills. Four terminals per school can provide 300 students a half hour CAI session each week and result in significant achievement gains.

When computer support for diagnosis and prescription is provided, students can receive significantly more individual attention from their teachers. In secondary schools utilizing the computer for management in geometry, class size may be increased with no loss in achievement.

Students in special education present a different kind of problem and require special consideration. However, it may be that computer-assisted instruction serves as a means by which the less able student can effectively organize his mathematical thinking. Every student who was pretested and posttested in this study showed a gain in arithmetic scores.

THE YEARS AHEAD

The CAI Program has acquired strong evidence that computers can have an important role in the instructional process. Although it is anticipated that CAI will be economically feasible for wide-spread utilization within 3-5 years, it must be understood that with the existing IBM 1500 CAI system or the proposed IBM 370 system that CAI is not currently economically feasible for all students. Therefore, the use of this technology should be limited to instruction for those students for whom the significant achievement gains justify the expenditure of extra dollars. To be specific, it is recommended that CAI be provided to those students who are achieving below grade level, to special education students, to those situations where increased class sizes can help displace hardware costs, or for computer education courses and problems solving which requires computer support.

Based upon the experiences of the last six years, results of evaluation studies and cost analysis, two major CAI Program thrusts are anticipated for the 1974-75 school year, elementary arithmetic and secondary mathematics.

ELEMENTARY ARITHMETIC

Validated CAI arithmetic packages will be provided to students achieving one or more years below grade level in 13 elementary schools. This will provide approximately 4,000 students with 30 minutes per week of CAI diagnosis and drill in operations with whole numbers, fractions, and percents.

The CAI arithmetic materials will be provided to underachieving students based upon the following predictions:

1. That the achievement of at least 90 percent of the underachieving students using the CAI programs will be at or above grade level in arithmetic within two years. This means that a child entering the fourth grade one or two grade levels behind, will enter the sixth grade at or above grade level in arithmetic skills. The total two-year cost per student will be $216. This amount will provide the student with CAI for a one-half hour period for each week for two years at a cost of $6 per hour which covers computer, staff, communications, and all other program costs.

   As computer equipment and instructional terminal costs are projected to decrease dramatically within the next five years, the cost for improved arithmetic achievement should be reduced from $216 to $72 or less per student for a similar two-year period by 1978.
2. That arithmetic achievement for students in special education will be substantially increased above expectancy. It is predicted that 80 percent of these students will achieve an increase in arithmetic skills of one grade level per school year. The cost for this achievement is $108 per student per year, which will provide one-half hour per week of CAI. By 1978, it is projected that similar results can be obtained for special education students for $36 per year.

The above projections are supported by evaluation data collected at the CAI Program. The 1971-72 sixth-grade study showed significantly greater achievement through CAI than by traditional instruction. A retention study conducted in the fall showed that these significant gains were maintained. In addition, using the Iowa Test of Basic Skills arithmetic scores given in October 1972, showed the CAI students with a mean grade score of 7.52, and the control students with a mean grade of 7.02. A 1972-73 fourth-grade study showed that CAI students made an average gain of 7.7 months in four months as compared with the control students mean gains of 4.5 months. An examination of the low halves showed the CAI group making a mean gain of 5.74 in raw score as compared with the control with a mean gain of 2.68 in raw score. All of the above CAI students received CAI 30 minutes per week.

During the past three school years, students from a special education high school have used the CAI arithmetic programs at Einstein High School. The average gain for these students during the 1971-72 school year was 7.6 months after 40-50 minutes of CAI use per week during a four-month period. During the 1972-73 school year, these students made a mean gain of one year in arithmetic achievement using CAI for an average of only 17.8 hours. National studies which have been conducted with mentally retarded adolescents show that achievement in the basic skills is difficult to maintain and increased achievement is rare. Special education students whose basic skills are improved will be able to perform simple clerical tasks and therefore increase the possibilities of their securing gainful employment.

SECONDARY MATHEMATICS

One computer-managed (CMI) and ten computer-assisted (CAI) instruction packages will be provided to 7 senior high schools beginning in September 1974. The two objectives for this action are:

1. To provide greater individualization and personalization with equal or greater achievement at potentially lower cost. Classes with computer management support may have 40 percent more students than classes without this technology; and
2. To increase achievement for students who are underachieving.

Computer support to the secondary mathematics program is based upon the following predictions:

—That computer-managed geometry classes can be individualized. Class size can be increased by 40 percent and each student will receive significantly more individual attention from his teachers than in traditional classes. Students will achieve as well or better than in traditional classes with average or underachieving students achieving above expectancy. Increasing the number of students in six classes will offset $7200 of the $18,000 program costs per school.

—With two nationally known computer hardware developers predicting a cost of 60¢-80¢ per terminal hour by 1978, the $18,000 terminal cost will be between $3600 and $4800 per year against a saving of $9000 in teachers salaries (assuming a conservative 5 percent per year salary increase). This would represent a net savings of between $700 and $900 per section of geometry per year.

CAI Program data supports the above statements on individual attention and overall achievement. A doctoral study by a MCPS administrator provided the information relative to average students in CAI classes achieving above expectancy.

SUMMARY

As in MCPS, other school systems that are active in CAI are moving forward and will be providing CAI exposure to greater and greater numbers of students. This trend marks a significant departure from the CAI activities of the last few years, and will provide expanded knowledge on the implementation planning necessary to achieve widespread effective utilization of CAI. The author feels that school use of CAI may, at least, be moving into the next phase of the implementation cycle.

Experimental data on page replacement algorithm

by N. A. OLIVER

General Motors Research Laboratories
Warren, Michigan

INTRODUCTION

Although paged VM (Virtual Memory) systems are being implemented more and more, their full capabilities have not yet been realized. Early research in this field pointed to possible inefficiencies in their implementation.[1-3] Subsequent studies, however, led to the conclusion that paged VM systems could provide a productive means to run large programs on small main memory, if proper techniques are employed.[4-7] One of the most influential of these is the choice of an efficient page replacement algorithm (RA) to minimize page traffic between the different levels of memory.

This paper compares the performance of two RAs about which little system performance measurement data is available. They are: the Global Least Recently Used (LRU) and the Local LRU with fixed and equal size main memory buffer allotted to each task. The number of page faults caused during execution of programs under each RA is used as an inverse criterion for its effectiveness.

These studies were conducted at the General Motors Research Laboratories (GMR) on the CDC STAR-1B* Virtual Memory computer[8] (core size=65K of 64 bit words; auxiliary/main memory access time ratio of 50000) with the Multi-Console-Time-Sharing (MCTS) operating system.[9]

PAGE REPLACEMENT ALGORITHMS

A basic problem in paged VM systems is deciding which page should be removed from main memory when an additional page of information is needed. Obviously, it should be a page with the least likelihood of being needed in the near future. Therefore a simple criterion for the "goodness" of a page RA is the minimization of page traffic between the main and auxiliary memories which is measured by the number of faults that occur during program execution.

One of the most popular page replacement strategies is LRU (Least Recently Used) strategy. The following RAs are based on it:

1. Global LRU RA: The replaced page is the one that has not been referenced for the longest period of real time, regardless of the task to which it belongs. This RA, which is a varying partitions RA by default, is heavily considered in literature.[5,10,11] Comparison results with various RAs obtained via simulation techniques and interpretive execution[1,12,13] are available. However, few if any (non-simulation) system measurements have been conducted. Existing (and fully developed) VM operating systems utilizing variations of this RA known to the author are: CP/67,[14,15] Multics,[14,16] MTS,[14] VS1[17] and VS2.[18]
2. Local LRU with fixed main memory paging buffer per task RA: The least recently used selection is made from pages belonging to the task which generated the page fault. Some treatment[1,4,10,13,19,20] and measurements of this RA were found in literature. However, only one operating system (besides the interim version of MCTS) implements a remote variation of this RA. It is the original IBM version of TSS.[14,21]
3. Local LRU with varying (working set)[5] partitions RA (WSRA): The replaced page is the least recently used page which does not belong to a working set of any task. Extensive literature is available.[4,5,6,7,11,13,19,20,22,23,24,25] Two true implementations (Burroughs B6700[26] and CP/67 at IRIA France[26]) and one approximation (the current version of TSS[27]) of this RA are known with limited measurement results.

Due to the implementation difficulties of the WSRA, only limited (special-case) measurements were taken of it.

THE TESTING ENVIRONMENT

System characteristics

Page-table: The STAR computer page-table[28] provides an address translation mechanism for all memory references. It points to pages of main memory in use and provides the mapping between the virtual address and the physical location of a page. The page-table ordering is hardware maintained; its entries (one for each page) are LRU ordered. Thus, the most recently accessed pages migrate to the top of the table while the least recently used move to the bottom. (The difference in address translation time between top and

* STAR-1B is a microprogrammed prototype version of the STAR-100 CDC computer.

TABLE I—Results of Identical-tasks Test

Customer tasks tested	Number of terminals	Global LRU (#P.F.)	Local LRU (#P.F.)	Local/Global LRU	Extreme paging buffer
Malus compilation of 185 source lines	1	88	88	1.000	
	2	245	488	1.991	23-68
	3	627	2241	3.574	
	4	3765	4674	1.241	
	5	6049	7863	1.299	
	6	9348	13901	1.487	
Malus compilation of 450 source lines	1	119	119	1.000	
	2	428	4078	9.528	18-73
	3	10474	11943	1.140	
OPL compilation of 160 source lines	1	86	86	1.000	
	2	133	242	1.819	30-61
	3	446	751	1.683	
OPL compilation of 575 source lines	1	143	143	1.000	
	2	382	917	2.400	24-67
INV matrix inversion 200×200	1	103	103	1.000	
	2	15524	15548	1.001	45-46
	3	15628	15688	1.001	
LIST_CAT sorting routine	1	104	104	1.000	
	2	116	120	1.034	43-48
	3	125	190	1.520	
PANICD dump formatting routine	1	448	448	1.000	
	2	461	477	1.035	44-47
	3	490	498	1.016	
	4	520	519	0.998	

bottom entries of the page table, due to longer search time, is insignificant relative to the other system time parameters.)

Level of multiprogramming: The MCTS operating system can be multiprogrammed up to a level corresponding to the maximum number of terminals supported by the system, which is seven.

Scheduling: A round robin scheduling scheme among the multiprogrammed tasks is employed. Tasks which are in page or other I/O wait state are skipped. When a task's time-slice expires that task is replaced by a task waiting for service and which is also chosen in a round robin fashion. If none is waiting the task with expired time-slice is allowed to continue. (In this study the level of multiprogramming is always equal to the number of running tasks and thus the time-slice parameter is not utilized.)

Paging space: It includes a maximum of 92 pages. Each page contains 512 64-bit words. On the interim MCTS system, this space is equally divided among the multiprogrammed tasks.

Paging mechanisms

In the interim version of MCTS, each task has a private page-table. Depending on the paging space, each multiprogrammed task is allotted a fixed number of pages. The Local LRU RA is used for page replacement.

For comparison purposes, MCTS was reprogrammed with the Global LRU RA. Only the paging mechanism was changed. No other system parameters such as multiprogramming, scheduling, paging space, etc., were modified.

The modifications for the Global LRU involved the use of a single page-table for all the customer tasks. All available pages in main memory were put into a general pool. When a page fault occurred, the Global LRU page which was the last entry in the hardware-managed single page-table, was replaced. No sharing of pages was allowed.

TESTING TECHNIQUES

In an effort to choose typical and diverse applications, these GMR developed customer tasks were tested:

Malus—A compiler for PL/I-like language designed to generate object code for the STAR computer. Two compilations, one of 185 and the other of 450 source lines were measured.

OPL—A compiler for computer graphics language. Again, two compilations, one of 160 and the other of 575 source lines were examined.

INV—A matrix inversion routine. Measurements were taken for inversion of a 200×200 order matrix.

LIST_CAT—A sorting routine. For this study a list of 700 names was sorted in several different orders.

PANICD—A compute-bound routine designed to format MCTS core dumps for printing. It formatted about 50000 words for these tests.

The tests were performed by running identical and nonidentical tasks simultaneously from a varying number of terminals. Each set of tests was executed twice, once with the Global LRU system and then repeated with the Local

LRU system. The total paging space was held constant in each case for both systems.

RESULTS

Identical-tasks test

Table I displays the average number of page faults per task (# P.F.) generated by identical multiprogrammed tasks which are running simultaneously on each of the two paging systems for varying number of terminals. Also included are ratio values representing the relative performance of the Local LRU in relation to the Global LRU. (The "extreme paging buffer" column will be explained later.)

While experimenting with the Global LRU system, it was observed that the number of pages used by each of the simultaneously running tasks varied considerably during execution. On the other hand the number of pages used by each task with the Local LRU system remained constant (by design). For example, in the two terminal Malus compilation of 450 source lines (Table I), which displays the most extreme difference, the Local LRU system divided the available 91 pages between the two tasks giving one 45 and the other 46 pages. With the Global LRU, on the other hand, tasks competed with each other for pages in main memory and the number of pages which each "owned" as a function of the elapsed execution time is displayed in Figure 1.

As these two identical tasks were started at the same instant, one would tend to think that they would split the core pages evenly among them and each would occupy close to half of memory at any given time (as in the Local LRU case). But as can be seen from Figure 1 this did not happen. These tasks dynamically changed the number of pages which they occupied with significant fluctuation.

One explanation might be that while executing, most programs change their locality[5] characteristics and consequently their working set size changes. If tasks are allowed to compete for pages, they tend to accumulate as many working set pages as they can in order to run effectively. For this purpose they use pages obtained from other tasks which at that moment (due to the negligible difference in starting time) are executing at other stages of the same program at which they usually have different locality properties and possibly require, or are forced to occupy, a smaller working set of pages.

In addition, it was observed that even though the above two tasks, started virtually at the same time, one task finished executing well ahead of the other. This can be clearly observed in Figure 1. At the start of execution Task #1 had 51 pages while Task #2 had only 40. Afterwards in most cases Task #2 had more pages. At point A Task #2 completed its execution and its pages started migrating to Task #1. At point B all of main memory belonged to Task #1. Due to this page migration from one task to the other and vice versa, Task #2 ran better up to point A and thus Task #1 was able to run efficiently from point B to completion.

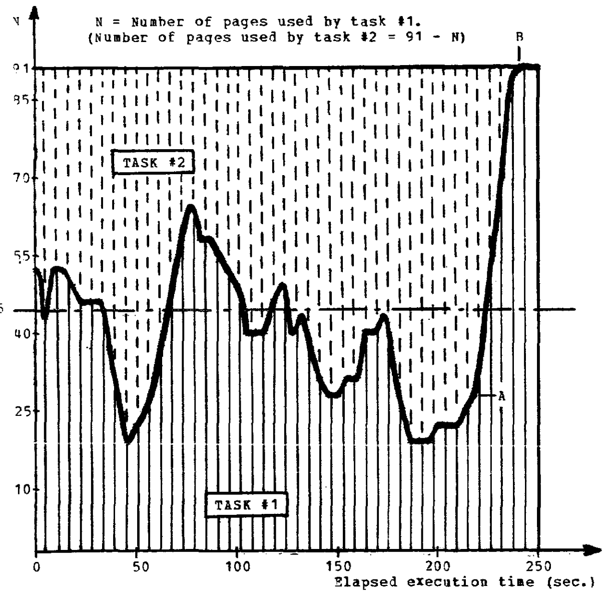


Figure 1—Variation in the number of pages "owned" by each of two tasks while executing under the Global LRU policy

The elapsed execution time with the Global LRU system was considerably shorter for high Local/Global ratios. But whenever the ratio was close to one, this time was virtually equal. As an example, in the case displayed in Figure 1 the compilation under the Global LRU system lasted only 231 seconds while under the Local LRU system the same compilation took 746 seconds.

The compilation cases generated considerably more page-faults under the Local LRU RA. But the difference in number of page-faults generated is less severe for LIST_CAT, and there is almost no difference in the INV and PANICD cases. This could be explained by the different working set characteristics of these programs. The Malus and OPL compilers change their working set sizes dynamically at a high rate while the rest of the tasks tend to have a fixed size or slowly changing working sets of pages. An indication of the rate of change of the working set size, in each case, for a multiprogramming level of two, can be obtained from the "extreme paging buffer" column in Table I. The figures in this column represent the number of pages each of the two tasks "owned" during the most extreme situation for that run with the Global LRU RA. As the extreme buffer difference gets larger, so does the performance ratio which is displayed in the adjacent column of Table I.

Non-identical-tasks test

At this point we felt that although the Local LRU showed, in some cases, poor performance when running the same tasks one against the other, it might still be a useful tool to

TABLE II— Results of Non-identical-tasks Test

Customer tasks tested	Global LRU (#P.F.)	Local LRU (#P.F.)	Local/Global LRU
Malus compilation of 450 source lines	1075	7239	6.734
OPL compilation of 575 source lines	1017	4271	4.200
INV matrix inversion 200× 200	1221	4822	3.949
PANICD dump formatting routine	519	557	1.073

prevent a complete system degradation in cases where some of the running tasks are in a thrashing state while others are not. We thought that by having a separate page table and a fixed number of pages for each task, the thrashing task would only degrade itself without affecting the rest of the system.

To test this situation, a four non-identical task mixture displayed in Table II was simultaneously run from four terminals. This experiment was designed so that all the tasks except PANICD would thrash with both the Global and Local LRU systems. Under the Local LRU every task "owned" one fourth of core while under the Global LRU all tasks compete for pages. Thus PANICD which does not thrash when running with one-fourth of core should have benefited from the "protection" provided to its paging buffer under the Local LRU algorithm whereas under the Global LRU the thrashing tasks could affect its performance by "taking away" its essential pages because they are in high need for pages.

But as can be seen in Table II, the number of page-faults which PANICD generated was not affected at all by the thrashing tasks.

The explanation to these unexpected results might be that tasks which are running effectively (PANICD), reference frequently (and thus "protect") their slow changing working set of pages. On the other hand the thrashing task needs many pages, each page for a short interval, and does not reference the same pages too often. Thus the pages of the thrashing task are, in most cases, the least recently used pages which migrate to the bottom of the page-table and are consequently overwritten. The thrashing task is only slightly affected by this process since chances are that it will need many other pages before requiring the pages which were just lost.

Working set replacement algorithm measurements

As we did not implement this RA which implies use of varying partitions with varying working-set sizes, we decided to test at least a special case of it, which is running a task with a fixed working set size in a fixed size partition.

PANICD has a small and fixed size working set of pages. (It contains the procedure pages, one input and one output data pages.) This fact explains the similar performance of the Local and Global LRU for PANICD as shown in Tables I and II. The slight difference in the number of page faults is attributed to the execution of some system programs known as "command language" before and after the actual execution of PANICD. These programs require large and rapidly changing working set sizes and thus contribute to the fewer number of page faults for the Global LRU in most cases.

In order to get relative performance measurements of the WSRA versus the Global LRU we decided to eliminate the effect of the "command language" by initiating the measurements only after all the multiprogrammed PANICD tasks have started their actual execution and terminate the data collection just before the first PANICD task branches back to the "command language." Thus a fixed size working set was required by each PANICD task at any time.

Since the WSRA requires that each multiprogrammed task have at least its working set of pages in main memory at all times, a Local LRU level of multiprogramming which provides a partition larger then the working set size will actually satisfy the WSRA requirements. The results for running identical PANICD tasks from a different number of terminals (corresponding to different levels of multiprogramming) are presented in Table III.

The column "dump units processed per second" presents the total number of dump-pages formatted by all the PANICD tasks, divided by the elapsed time required to run all the tasks at each multiprogramming level. Thus this column represents the real throughput of the entire system. The other column "number of page faults per dump unit processed" shows the total number of page faults generated by all tasks, divided by the total of all the dump-pages formatted.

The throughput of the system increases with the level of multiprogramming, for both systems, up to the level of three and four while the number of page faults per unit dump remains low. Beyond the level of four both systems are thrashing and the throughput consequently deteriorates. Thus under the WSRA policy we would have run the system

TABLE III—System Throughput and Page Fault Frequency for Increasing Levels of Multiprogramming

Number of terminals (multiprog. level)	Global LRU		Local LRU (WSRA)	
	Dump units proc./sec.	#PF/dump units proc.	Dump units proc./sec.	#PF/dump units proc.
1	0.37	4.21	0.37	4.21
2	0.42	4.22	0.42	4.22
3	0.46	4.23	0.46	4.23
4	0.46	4.24	0.46	4.23
5	0.42	7.55	0.42	7.82
6	0.39	11.90	0.40	10.19
7	0.29	20.30	0.31	21.10

at a multiprogramming level not higher than four. But for levels one through four the Global and Local LRU, which in this case is identical to the WSRA, perform virtually the same.

Above the level of four the Local LRU (WSRA) does show better performance and that is probably due to the fact that with the Global LRU system a task which is waiting the longest on the scheduler queue and is the next one to run its pages become the least recently used ones and are overwritten. As this happens only after reaching a thrashing level of multiprogramming, a Global LRU RA can be useful only if the system can detect when overloading occurs. Such a performance monitor (based, for example, on the page-faulting level of the entire system) could be used to reduce the multiprogramming level whenever thrashing occurs to prevent performance degradation.

CONCLUSIONS

The results of this study strongly indicate that artificially restricting the main memory space which a task may utilize in a paged VM system results in an increased page traffic between the different levels of memory and consequently in considerable loss of efficiency. Tasks, especially if they require rapidly changing working set sizes, should be allowed to compete freely for the space which each may occupy at any given time.

The Global LRU RA performed better than the Local LRU RA with fixed partitions and matched the performance of the Local LRU with varying partitions (WSRA), for a non-thrashing situation, in this study. The following Global LRU virtues should be noted:

1. It is a simple varying partitions RA in which the partition size is controlled by the RA itself and not by the operating system.
2. It is highly unlikely that thrashing tasks can "overtake" main memory and thus "hurt" the performance of non-thrashing tasks. This is due to the fact that non-thrashing tasks reference frequently and thus "protect" their essential pages from becoming the LRU ones. On the other hand the thrashing task needs many pages, each for a short interval, and does not reference the same pages too often. Thus the pages of the thrashing task are, in most cases, the LRU ones and are consequently overwritten.
3. Critics of the Global LRU strategy (including the author[7]) claim that with the Global LRU RA, the task which is idle for the longest time while waiting on the scheduler queue, and which is the next to run, is most likely to find its pages missing. Evidence of this has been found in these studies. But it turns out that the space could be utilized more effectively by the current running task than it would have been if these pages had been reserved without utilization for the delayed task.
4. In addition to the performance advantage (reduction of execution time and number of page faults), the Global LRU with the single page-table is easier to manage and requires less operating system space than the Local LRU with multi-page-tables and fixed paging buffer system.

The Global LRU algorithm is especially useful for simple, round robin scheduled operating systems. For more sophisticated systems, however, the multi-page-table approach might be useful due to requirements other than efficiency, such as: priority scheduling, etc. But since no artificial restrictions should be imposed on main memory space it seems that the working-set-partitions RA's such as the WSRA[5] and PFF[13] might well be the only class of paging strategies able to perform effectively utilizing multi-page-tables. However, the WSRA will require a "smart" mechanism to determine the following: (A) The size of the task's working set at any given time. (B) When a task is in a working set expansion phase and needs more pages, which of the other multiprogrammed tasks will be the one to lose pages. (C) What action should be taken when there are a few available pages in core but not enough to start a new task. In the Global LRU case no information about (A) is needed; the decision about (B) is trivial; as for (C) the new task is started and it "fights" to build its working set from pages which are probably non-useful to other tasks.

The WSRA has a clear advantage over the Global LRU; It prevents system overloading. Thus if the Global LRU is to be used, a special system performance monitor could be used to reduce the level of multiprogramming whenever overloading occurs.

ACKNOWLEDGMENT

I wish to thank G. G. Dodd for his support of this study and advice on organizing the paper. Also I am grateful to R. R. Brown, J. W. Boyse, M. Cianciolo and the rest of the MCTS personnel for their cooperation. Last but not least I wish to thank P. J. Denning and W. W. Chu for their constructive criticism of this paper.

REFERENCES

1. Coffman, E. G. and L. C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment," *Comm. ACM*, 11, July 1968, 471-474.
2. Fine, G. H., C. W. Jackson and P. V. McIsaac, "Dynamic Program Behavior under Paging," *Proc. 21st Nat. Conf. ACM*, ACM Pub. P-66, 1966, pp. 223-228.
3. Kuehner, C. J. and B. Randell, "Demand Paging in Prospective," *Proc. AFIPS 1968 Fall Joint Comp. Conf.*, Vol. 33, pp. 1011-1018.
4. Denning, P. J., "Virtual Memory," *Computing Surveys*, Vol. 2, No. 3, Sept. 1970, pp. 153-189.
5. Denning, P. J., "The working-set model for program behavior," *Comm. ACM.*, Vol. 11, May 1968, pp. 323-333.
6. Chu, W. W., N. Oliver, and H. Opderbeck, "Measurement Data on the Working Set Replacement Algorithm and Their Applica-

tions," *Proc. Brooklyn Polytechnic Institute Symposium on Computer-Communications Networks and Teletraphic*, Vol. 22, Apr. 1972.

7. Oliver, N., *Optimization of Virtual Paged Memories*, Master thesis, Univ. of Calif. Los Angeles, 1971.
8. Holland, S. A., and C. L. Purcel, "The CDC STAR-100 a large scale network oriented computer system," *IEEE Proc. of the International Computer Society Conference*, Boston, Mass., Sep. 22-24, 1971.
9. Brown, R. R., J. L. Elshoff, and M. R. Ward, et al., *Collection of MCTS Papers*, to be published, G. M. Res. Labs., Warren, Mich. 1974.
10. Belady, L. A., "A Study of Replacement Algorithms for a Virtual-storage Computer," *IBM Syst. J.*, Vol. 5 No. 2, 1966, pp. 78-101.
11. Denning, P. J., "Thrashing: Its Causes and Prevention," Proc. *AFIPS 1968 Fall Joint Comp. Conf.*, Vol. 33, pp. 915-922.
12. Thorington, J. M., J. D. Irvin, "An Adaptive Replacement Algorithm for Paged-memory Computer Systems," *IEEE Trans.* Vol. c-21, Oct. 1972, pp. 1053-1061.
13. Chu, W. W. and H. Opderbeck, "The Page Fault Frequency Replacement Algorithm," *Proc. AFIPS 1972 FJCC*, Vol. 41, pp. 597-609.
14. Alexander, M. T., *Time Sharing Supervisor Program*, Univ. of Mich. Computing Center, May 1969.
15. Bayels, R. A., et al., *Control Program-67/Cambridge Monitor System (CP-67/CMS)*, Program Number 360D 05.2.005, Cambridge, Mass., 1968.
16. Organick, E. I., *A Guide to Multics for Sub-System Writers*, Project MAC, 1969.
17. *IBM OS/Virtual Storage 1 Features Supplement*, No. GC20-1752-0.
18. *IBM OS/Virtual Storage 2 Features Supplement*, No. GC20-1753-0.
19. Coffman, E. G. and T. A. Ryan, "A Study of Storage Partitioning Using a Mathematical Model of Locality," *Comm. ACM 15*, March 1972, pp. 185-190.
20. Oden, P. H. and G. S. Shedler, *A Model of Memory Contention in a Paging Machine*, IBM Res. Tech. Rep. RC3056, IBM Yorktown Heights, N. Y., Sept. 1970.
21. *IBM System/360 Time Sharing Operating System Program Logic Manual*, File No. S360-36 GY28-2009-2, New York 1970.
22. Denning, P. J. and S. C. Schwartz, "Properties of the Working-Set Model," *ACM*, 15, March 1972, pp. 191-198.
23. DeMeis, W. M. and N. Weizer, "Measurement Data Analysis of a Demand Paging Time Sharing System," *ACM Proc.* 1969, pp. 201-216.
24. Openheimer, G. and N. Weizer, "Resource Management for a Medium Scale Time-sharing System," *Comm. ACM*, Vol. 11, May, 1968, pp. 313-322.
25. Spirn, J. R. and P. J. Denning, "Experiments with Program Locality," *Proc. AFIPS 1972 FJCC*, Vol. 41, pp. 611-621.
26. Private communication with P. J. Denning.
27. Doherty, W. J., "Scheduling TSS/360 for Responsiveness," *Proc. AFIPS 1970 FJCC*, Vol. 37, AFIPS Press, Montvale, N. J., pp. 97-112.
28. Curtis, R. L., "Management of High Speed Memory in the STAR-100 Computer," *IEEE Proc. of the International Computer Society Conference*, Boston, Mass., Sep. 22-24, 1971.

Some programming techniques for processing multi-dimensional matrices in a paging environment

by JAMES L. ELSHOFF
General Motors Research Laboratories
Warren, Michigan

INTRODUCTION

Although virtual memory systems are supposed to free the programmer from space management problems, the systems do not always succeed. In fact, programmers find that by ignoring the fact that real core is limited, the cost of executing their programs sometimes makes them unusable, not to mention some of the detrimental effects the program has on the throughput of the overall system. This problem seems to be especially prevalent when large matrices of data are involved. The data are usually referenced in a cyclical pattern and when the entire matrix will not fit in core, the number of page faults encountered during execution is maximized. The focus of this paper is to analyze programming techniques which will reduce the number of page faults in matrix operations and thereby improve program performance.

Program behavior in a paging environment has been studied[1,2,3,4] from several points of view. Specifically, Brawn, Gustavson, and Mankin[5] have concerned themselves with processing vectors in a paging environment. Moler and Dubrulle[6,7] have looked at two separate matrix operations with respect to execution in a virtual memory environment. Several storage schemes and related operations for matrices were analyzed with respect to paging systems by McKellar and Coffman.[8] Also, Guertin[9] presented some programming examples to improve program behavior in a demand paging system.

The work presented in this paper was done on the premise that a programmer must be aware of how his program will reference the data during execution. The programmer will not be completely free of space management considerations in the design of his algorithms. The material presented deals with the mathematically simple problems of matrix addition, transposition, and multiplication. The methods of problem analysis and the programming guidelines are intended to give the working programmer new tools for doing a better job.

THE WORKING ENVIRONMENT

Although the material presented in this paper is directly extendable to matrices with more than two dimensions, the matrices used in the examples will all be two-dimensional for the sake of simplicity. The indices will refer to the row and column of the matrix from left to right. An $M\times N$ matrix A will have M rows numbered 1 through M and N columns numbered 1 through N. All matrices will be assumed to be stored rowwise. A 2×2 array A will have its elements mapped into linear virtual memory space in the order A(1,1), A(1,2), A(2,1), A(2,2). The order of the subscripts within the subscript list may be reversed throughout this paper for columnwise storage.

The paging algorithm executed by the operating system will be the least recently used LRU algorithm.[2] This algorithm was chosen because most operating systems available either use this algorithm or an approximation to it. Also, the experimental results shown in the latter sections of the report were generated on a computer with LRU hardware. Note that this algorithm is used as a basis for the derivation of formulas and is not essential to the premises upon which this paper is founded.

The total number of page faults processed during a complete matrix operation will be used as a measure of the program performance. The CPU time required to perform the matrix operation will be considered to be constant. Implementation of some of the programming techniques described herein may increase program execution time due to additional loop controls, but this is considered to be negligible because the additional CPU time is measured in microseconds while the time to process a page fault is measured in tens of milliseconds.

The examples used are coded using PL/I DO statements for conciseness and readability. Except for matrix mapping functions, the programs being considered are really language independent. FORTRAN programmers may have to use IF loops instead of DO loops since they cannot specify negative increments on their DO statements. Furthermore, the programs shown are not written to minimize CPU time. Overlaying each two-dimensional matrix with a vector is an obvious method of reducing CPU time.

Table I lists some of the symbols and their respective meanings which are used throughout this paper. The notation $\lceil a \rceil$ will be used to signify the smallest integer greater than or equal to a and $\lfloor a \rfloor$ will signify the largest integer less than or equal to a.

In order to facilitate analysis, the first element of an array will be stored as the first word in a page. The matrix dimensions will satisfy the inequalities $N \leq S < N^2$. That is, at least one row of the matrix will fit in a page but not the whole matrix. The programming techniques hold when $S < N$ but the formulas derived will not. In order to make the problem of interest, $k < \sum p_i$ is also assumed. Furthermore, the executing code and the temporary variables are resident in real memory.

THREE PROGRAMMING TECHNIQUES

In this section three programming rules will be described which can be applied to multi-dimensional array operations in order to improve program performance. The circumstances under which each may be applied and the benefits that may be expected are presented.

Ordering nested loops

Let A be an $M \times N$ matrix and B be an N element vector. Write a program so that each element of B contains the sum of all of the elements in the corresponding column of A. $b_i = \sum_i a_{ji}$. Each element of B is initialized to zero.

An obvious solution to this problem is to write a loop which will sum each column. Then enclose the loop in a second loop which will traverse all the columns.

```
DO COL = 1 TO N BY 1; /* Traverse each column */
   DO ROW = 1 TO M BY 1;
                          /* Sum a column       */
      B(COL) = B(COL) + A(ROW,COL);
   END;
END;
```

Now consider the reference pattern on matrix A, A(1,1), A(2,1), A(3,1), . . . , which causes each page spanned by matrix A to be referenced on each pass through the outer loop. By interchanging the DO statements the elements in matrix A will be referenced in the order in which they are stored. Thus, all of the elements in a single page will be processed while the page is in core. Furthermore, each page is only required to be in core once.

```
DO ROW = 1 TO M BY 1; /* Traverse each row    */
   DO COL = 1 TO N BY 1;
                          /* Add all row elements */
      B(COL) = B(COL) + A(ROW,COL);
   END;
END;
```

The minimum number of page faults $F_{min} = p_A + p_B$, since each page of both matrices must be brought into core at least once. For any number of real memory pages k, where $2 \leq k < F_{min}$, the first program will have $F = Np_A + p_B$ page faults while the second program will have $F = F_{min}$ page faults. Certainly the first program is related to N^2 in this case because of the LRU paging algorithm that is assumed. But even considering an optimum paging algorithm, the number of page faults $F = (k-1) + N(p_A - k + 2)$, where $p_B = 1$, page faults is the best that can be done. Thus, even when $k = p_A$, an optimum paging algorithm cannot get $F = F_{min}$ in the case of the first program.

TABLE I—Definitions of Symbols

Symbol	Meaning
A,B,C	Name of a matrix
F	Number of page faults
L,M,N	Dimension of a matrix
p_A	Number of pages spanned by matrix A
k	Number of pages of real (core) memory available for data
r_A	Number of rows of matrix A in one page
S	Page size in words (matrix elements)
q_i	Number of rows processed per pass through loop indexed by i

Rule 1. Nest loops so that the innermost loop defines the subscript with the minimum distance between elements when all other subscripts are held constant.

Rule 1 is a further generalization of a rule published by Guertin.[9] The rule may be applied iteratively to determine the second from innermost loop once the innermost loop is fixed, etc. The rule applies to most cases where nested loops are encountered and should always be considered by the programmer. In order to apply the rule, the programmer must understand the storage mapping algorithm of the language being used as well as the problem being solved.

Guertin discusses many variations and applications of Rule 1. One of the variations is paraphrased here as an example. Consider the following program which is like the first except that no initialization of the vector B is assumed.

```
DO COL = 1 TO N BY 1;
   B(COL) = 0;                    /* Initialize B */
   DO ROW = 1 TO N BY 1;
      B(COL) = B(COL) + A(ROW,COL);
   END;
END;
```

If the nested loops are interchanged, the program will no longer execute properly. But B can be initialized to zero in a separate loop at only a small additional cost. Better yet, initialize B with the first row of A and regain the additional loop time by eliminating M additions.

```
DO COL = 1 TO N BY 1;
   B(COL) = A(1,COL);             /* Initialize B */
END;
DO ROW = 2 TO M BY 1;
   DO COL = 1 TO N BY 1;
      B(COL) = B(COL) + A(ROW,COL);
   END;
END;
```

Processing multiple rows

Let A and B be N×N matrices and write a program to transpose B into A, $a_{ij}=b_{ji}$.

```
DO ROW = 1 TO N BY 1;
    DO COL = 1 TO N BY 1;
        A(ROW,COL) = B(COL,ROW);
    END;
END;
```

Rule 1 cannot be applied to this problem. Either A or B will have each of its pages referenced for each pass through the outer loop.

The number of page faults encountered in performing this transpose operation is $F=p_A+Np_B$ in the working environment that has been described. Obviously the product Np_B causes the number of page faults to be high. N is the number of rows in the A matrix. If A has more than one row in a page, why not process all of the rows at one time? Let the rows in one page r_A be represented by the variable RPP and the following program results.

```
DO ROWS = 1 TO N BY RPP;
                        /* Page of rows          */
    MAXROW = MAX(ROWS + RPP-1,N);
    DO   COL = 1 TO N BY 1;
    DO  ROW = ROWS TO MAXROW BY 1;
                        /* Each row in the page */
            A(ROW,COL) = B(COL,ROW);
        END;
    END;
END;
```

The number of page faults is reduced to $F=p_A+\lceil N/r_A\rceil p_B$ by processing multiple rows. For a small r_A, p_A and p_B are very large and the reduction is significant. For a larger r_A, the $\lceil N/r_a\rceil$ factor is much smaller and the reduction is still significant.

A reasonable question to ask is why not process $2r_A$ rows during each pass through the outer loop. In this problem the number of page faults would be halved. But what if $k=2$? Then the second factor is halved but the first factor becomes $2\lceil N/r_A\rceil p_A$ and the expected gain becomes a loss. Processing multiple rows reduces the page faults as long as all of the rows being processed remain in real memory. Consider that letting RPP=N in the above example is tantamount to inverting the DO statements in the original program.

Rule 2. Process all the elements in a page which vary with respect to the subscript causing maximum paging while that page is in core and the other subscripts are held constant.

Rule 2 generally applies to those problems for which Rule 1 cannot minimize the page faults. Such problems can be characterized in several ways. (1) The same index may be used in different subscript positions within the loop. (2) A single element is referenced more than one time during the course of execution. (3) An extensive calculation is performed within the loop which results in insufficient real memory pages even after applying Rule 1.

There is no reason why Rule 2 cannot be applied to more than one index. McKellar and Coffman[8] describe a matrix storage scheme which lends itself to matrix operations which apply Rule 2 to every subscript. For the storage scheme considered within this paper, Rule 2 has less effect on subscripts toward the right in the subscript lists since there are fewer elements per page which vary only in the right-hand subscripts.

Alternating matrix traversal direction

Consider the transpose problem used in the last section. The total number of page faults $F=p_A+Np_B$ was reduced by attacking the factor N, the number of rows in the A matrix. Another approach is to reduce the factor p_B, the number of pages spanned by the B matrix. Since B is too large to fit in real memory, and since the algorithm references B in a cyclic manner, each page of B is removed from real memory between references. Thus, p_B can be effectively reduced by referencing an arbitrary page more than once while the page is in real memory. The cyclic reference pattern is broken in order to accomplish this end.

```
COLSTART = 1;
COLEND = N;
COLDIFF = 1;
DO ROW = 1 TO N BY 1;
    DO COL = COLSTART TO COLEND BY
      COLDIFF;
        A(ROW,COL) = B(COL,ROW);
    END;
    TEMP = COLSTART;
    COLSTART = COLEND;
    COLEND = TEMP;
    COLDIFF = -COLDIFF;
END;
```

The new program references the B matrix by going down the first column, B(1,1), B(2,1), . . . , B(N,1), and up the second column, B(N,2), B(N−1,2), . . . , B(1,2). The program continues to alternate the direction of column traversal until the matrix operation is complete. Since the pages referenced near the end of one column traversal are the same as those referenced at the beginning of the following column traversal, a number of page faults may be eliminated. With the LRU paging algorithm that has been assumed, $F=p_A+p_B+(N-1)(p_B-k+1)$. Thus, for each of N−1 column traversals the number of page faults that can be eliminated is the number of real memory pages available for the B matrix.

Rule 3. Let the increment of a faster varying subscript alternate its sign each time a more slowly varying subscript changes, when the more slowly varying subscript appears to the right of the faster varying subscript.

Rule 3, like Rule 2, should be applied after Rule 1. Rule 3 is based on a paging algorithm which keeps the most recently referenced pages in real memory. Although few paging systems have a true LRU paging algorithm, most systems do approximate the LRU algorithm. Consequently, benefits may not always be as great as indicated here, but results will be significant. Rule 3 produces positive results when applied to all subscripts except the leftmost. Rule 3 will not generally apply to the outermost loop of a set of nested loops.

Summary of rules

Three rules have been given which may be used in order to decrease the total number of page faults encountered while performing matrix operations. Each rule attacks the problem from a different point of view and requires different knowledge on behalf of the programmer. Rule 1 eliminates page faults by aligning the reference pattern for the matrix elements with the storage mapping function; the programmer must know the storage mapping function. Rule 2 uses the programmer's knowledge of page size in the computer system in order to break a large problem into a series of smaller problems which generate fewer page faults. Finally, Rule 3 assumes a paging algorithm which approximates an LRU algorithm in order to reduce the number of page faults; consequently, the programmer must learn something about the paging strategy in the system.

THE TRANSPOSE OPERATION

An in-place matrix transpose operation is now analyzed with respect to the programming rules just given. The standard algorithm appearing in print

```
DO ROW = 1 TO N-1 BY 1;
    DO COL = ROW TO N BY 1;
        TEMP = A(ROW,COL);
        A(ROW,COL) = A(COL,ROW);
        A(COL,ROW) = TEMP;
    END;
END;
```

will cause p_A page faults in processing the first row. For each additional row in the first page, p_A-1 page faults will be incurred assuming the first page remains in core. When all processing has been completed on the first page, the rows on the second page will cause page faults on the remaining pages of the matrix in a similar manner. Finally, a point is reached where the remaining portion of the matrix will fit in real memory. The total number of page faults is given by

$$F=p_A+(r_A-1)(p_A-1)+(p_A-1)$$
$$+(r_A-2)(p_A-2)+\cdots+(k-1).$$

Applying the algebra pertaining to arithmetic progressions, the summation reduces to

$$F=p_A+\frac{r_A}{2}(p_A^2-p_A-k^2+k).$$

Two formulas were developed for analysis of the in-place transpose algorithm when multiple rows were processed each time through the outer loop. The first formula applies to the case where $r_A>q$, all of the rows fit within one page.

$$F=p+\frac{1}{2}\left(\frac{\lceil r_A-q\rceil}{q}+1\right)(p_A^2-p_A-k^2+k).$$

The second formula is derived from the case where the multiple rows being processed span an integral number of pages, $ar=q$, where a is a positive integer.

$$F=\frac{1}{2}\left(\frac{\lceil p_A-k\rceil}{a}+1\right)\left(2p_A-\frac{\lceil p_A-k\rceil}{a}a\right)$$

This formula also assumes that all of the multiple rows being processed will remain in real memory, $a<k$.

The number of page faults is given by

$$F=k+\left(p_A-k-\frac{\lceil p_A-k\rceil}{2}+2\right)\left(2r\frac{\lceil p_A-k\rceil}{2}-1\right)$$

when the matrix operation alternates the direction of loop traversal in order to reuse the matrix pages in real memory. All pages are assumed to be full in this case.

Finally, both the multiple row rule and the alternating direction rule can be applied in the same program. This combination results in the number of page faults being reduced to

$$F=p_A+\frac{1}{2}\frac{\lceil p_A-k\rceil}{a}\left[2(p_A-k)-a\left(\frac{\lceil p_A-k\rceil}{a}-1\right)\right]$$

where a is a positive integer such that $ar=q$ and $a<k$.

Figure 1 shows the number of page faults generated by each program as a function of real memory size. A matrix which spans 20 pages, $p_A=20$, is assumed. The number of rows processed at one time is equal to the number of rows in a single page, in this case $r_A=5$ and $a=1$. As larger matrices are considered, the curves maintain their relative position but the spread between them becomes greater.

Programs were written which applied the programming rules in the manner described in order to validate the expected results. A 101×101 matrix was transposed in place. A 512 word page contained 5 rows plus 7 elements. The matrix spanned 20 pages less 39 words. Five rows were processed at a time when the multiple row rule was used. An LRU paging algorithm was applied by the operating system. These tests were run on a dedicated machine with no inter-

TABLE II—Summary of Transpose Operation Data

Algorithm	Paging Buffer Size (k)	Expected Page Faults	Measured Page Faults	Measured / Expected
Standard	15	445	450	1.011
Multiple Rows	14	119	148	1.243
Alternating Direction	15	131	144	1.099
Combination	15	35	64	1.828

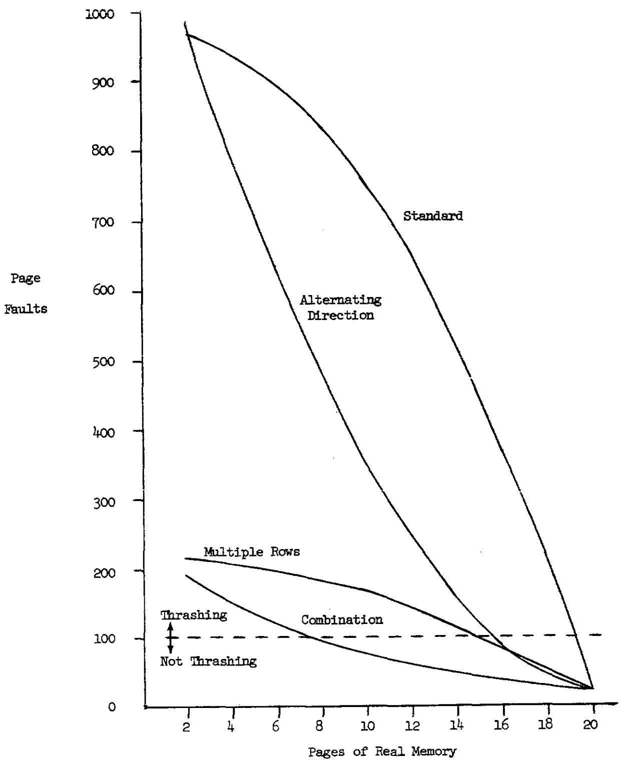


Figure 1—Comparison of transposition algorithms for a matrix spanning twenty pages

ference. The parameters, which did not fall within the constraints under which the formulas were derived, were (1) the last page spanned by the matrix was not full and (2) a page did not exactly contain an integral number of rows. Table II summarizes the results from these runs.

Additional statements had to be added to the standard program when the programming rules were applied. Table III shows the additional CPU time required by the new programs and also the less total CPU time, memory, and channel resources that are needed.

An arbitrary thrashing level was indicated by a superimposed line on Figure 1. The line represents approximately one page fault each 10 milliseconds in its present location. This line is of special interest since the distance between the line and a curve above the line is directly proportional to the length of time the program will thrash. The duration of such detrimental effects is shown in Table III. The duration would be even worse if this program were a customer in a time-sharing system where the program would be regularly removed from real memory. Thus, the 35 percent increase in CPU time for the transpose operation shown in Table III is negligible when compared to the overall reduction of 76 percent in total CPU time and 86 percent in elapsed time.

Another experiment was performed with the transpose operation. The purpose was to determine the validity of the formula for the multiple row transpose operation since the $\lceil r_A - q \rceil / q$ factor and the $\lceil p_A - k \rceil / a$ factor introduce error when the result of the division is not an integer. Figure 2 shows the error introduced. The error for the formula with the $\lceil p_A - k \rceil / a$ factor is nearly constant and is attributable to

TABLE III—Transpose Operation Resource Utilization

Algorithm	Problem CPU	System CPU	Total CPU	Elapsed Time	I/O Time
Standard	.819	9.900	10.719	77.5	66.8
Multiple Rows	.985	3.256	4.241	26.2	22.0
Alternating Direction	.917	3.168	4.085	25.5	21.3
Combination	1.110	1.408	2.518	11.0	8.5

All times are in seconds.

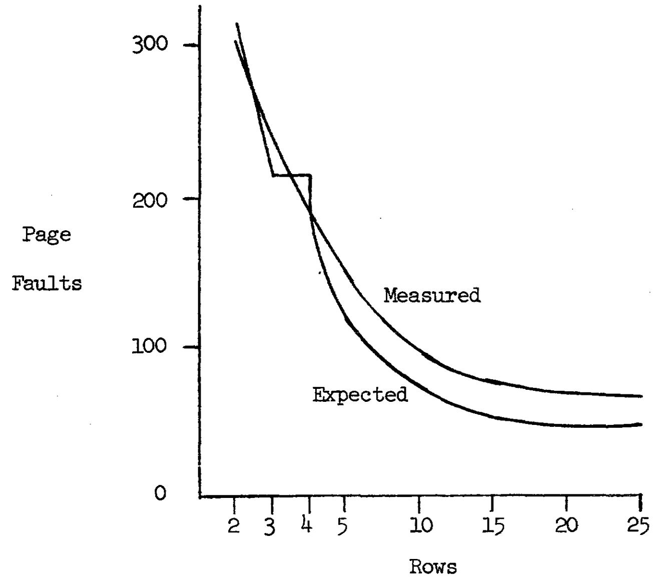


Figure 2—Error in the multiple row transpose operation

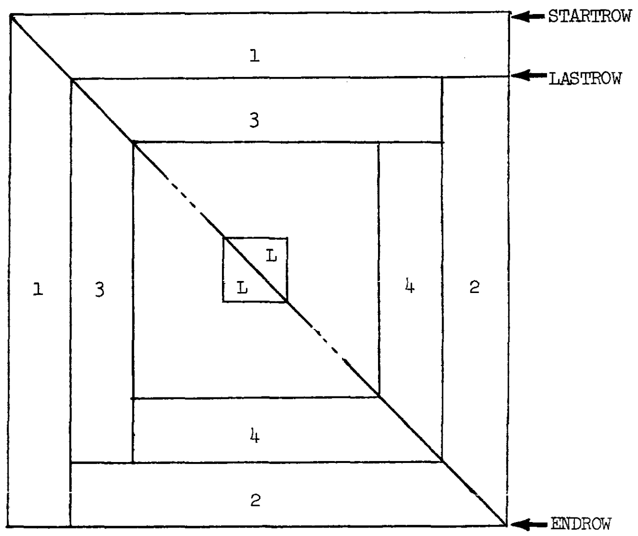


Figure 3—Transpose reference pattern using combination of rules

the fact that a page does not contain a integral number of rows. No attempt will be made to apply this result to other multiple row formulas; however, please note that error bounds are easily calculated on each of these formulas if such a comparison is desired.

A program which will perform an in-place transpose with both the multiple row rule and the alternating direction rule applied is shown as Example 1. Although the program may seem complicated initially, it really is easily understood. Figure 3 shows the manner in which the array is referenced with respect to the outer loop. All of the elements in the areas shown with a 1 are interchanged during the first pass through the DO WHILE loop. The second pass interchanges the elements in the areas marked with a 2. The operation continues until the elements in the areas marked L are interchanged. The variable names from the program shown in Figure 3 relate to the first pass through the DO WHILE loop.

```
STARTROW = 1;
ENDROW = N;
DIFF = 1;
DO WHILE ((STARTROW-ENDROW)*DIFF < 0);
    IF DIFF > 0 /* DETERMINE LAST ROW FOR
      THIS LOOP */
    THEN LASTROW = MIN (STARTROW +
      RPP * DIFF,ENDROW) - DIFF;
    ELSE LASTROW = MAX (STARTROW +
      RPP * DIFF,ENDROW) - DIFF;
    DO COL = STARTROW + DIFF TO LASTROW
      BY DIFF;
        IF DIFF > 0 /* DO NOT CROSS
          DIAGONAL */
        THEN TLASTROW = MIN(COL-DIFF,
          MAXROW);
        ELSE TLASTROW = MAX(COL-DIFF,
          MAXROW);
        DO ROW = STARTROW TO TLASTROW
          BY DIFF;
            TEMP = A(ROW,COL);
            A(ROW,COL) = A(COL,ROW);
            A(COL,ROW) = TEMP;
        END;
    END;
    TEMP = STARTROW; /* ALTER DIREC-
      TION */
    STARTROW = ENDROW;
    ENDROW = TEMP + RPP * DIFF;
    DIFF = -DIFF;
END;
```

Example 1—In Place Transpose Processing Multiple Rows and Alternating Directions

MATRIX MULTIPLICATION

In order to further investigate the programming rules given earlier, consider a simple program for performing matrix multiplication. Let A, B, and C be L×M, M×N, and L×N matrices, respectively. A program which computes C=A*B is shown.

```
DO ROW = 1 TO L BY 1;
    DO COL = 1 TO N BY 1;
        C(ROW,COL) = 0;
        DO INNER = 1 TO M BY 1;
            C(ROW,COL) = C(ROW,COL) +
              A(ROW,INNER) * B(INNER,COL);
        END;
    END;
END;
```

Whenever this standard matrix multiply program is executed, at least one page of both matrix A and matrix C and all of matrix B must be resident in core in order to get reasonable performance; otherwise, each page of B is going to cause LN page faults. In fact, for any amount of real memory pages k, where $p_B+2>k>3$, the number of page faults encountered by the above program is given by

$$F=p_A+LNp_B+p_C.$$

According to rowwise storage scheme that has been assumed, the DO statements with the ROW and COL indices are in the proper order. The B matrix is referenced in columnwise fashion while the A and C matrices are referenced in a rowwise fashion. If the two DO statements in question are interchanged, the number of page faults increases to

$$F=Mp_A+LNp_B+Mp_C.$$

By the reordering rule the DO statements with the COL and INNER indices should be interchanged if possible. The index COL never appears in any subscript position but the rightmost; therefore, the DO statement controlling the COL index should be innermost. Furthermore, the INNER and COL indices are only used together when the B matrix is referenced, and INNER is to the left of COL in that case. The number of page faults is reduced to

$$F=p_A+Lp_B+p_C$$

TABLE IV—Matrix Multiply Page Fault Formulas

Rule 1	Rule 2	Rule 3	Page Fault (F)
	X		$p_A+\frac{\lceil L\rceil}{q}Np_B+p_C$
		X	$p_A+k-2+LN(p_B-k+2)+p_C$
X	X		$p_A+\frac{\lceil L\rceil}{q}p_B+p_C$
X		X	$p_A+k-2+L(p_B-k+2)+p_C$
	X	X	$p_A+k-a-c+\frac{\lceil L\rceil}{q}N(p_B-k+a+c)+p_C$
X	X	X	$p_A+k-a-c+\frac{\lceil L\rceil}{q}(p_B-k+a+c)+p_C$

by interchanging the DO statements and adding an appropriate loop to initialize the C matrix.

The DO statements controlling the ROW and INNER indices should not be interchanged since ROW is always in the leftmost position. Calculating the number of expected page faults verifies this fact, since interchanging the DO statements results in more page faults.

$$F = Mp_A + LMp_B + p_C$$

The multiple row rule and alternating direction rule may also be applied to the matrix multiplication operation. Table IV shows the formulas for the number of page faults expected when different combinations of the rules are applied. The variables a and c represent the number of pages required to hold q rows of the A and C matrices respectively, where q is the number of rows being processed during each pass through the outermost loop. All of the formulas only hold for the condition $a + c < k$; that is, all the rows of A and C being processed during one outer loop traversal remain in real memory.

Each of these formulas has been evaluated for a matrix multiplication. Each matrix is assumed to be 101×101, $L = M = N = 101$, spanning 20 pages, $p_A = p_B = p_C = 20$. The multiple row algorithms will process five rows at a time, $q = 5$ and $a = c = 1$. The amount of real memory varies. The results are displayed on the semi-logarithmic graph in Figure 4. For matrix multiplication the number of page faults can be reduced from over 200,000 to near 100 by application of the programming techniques described.

The matrix multiplication algorithms were programmed

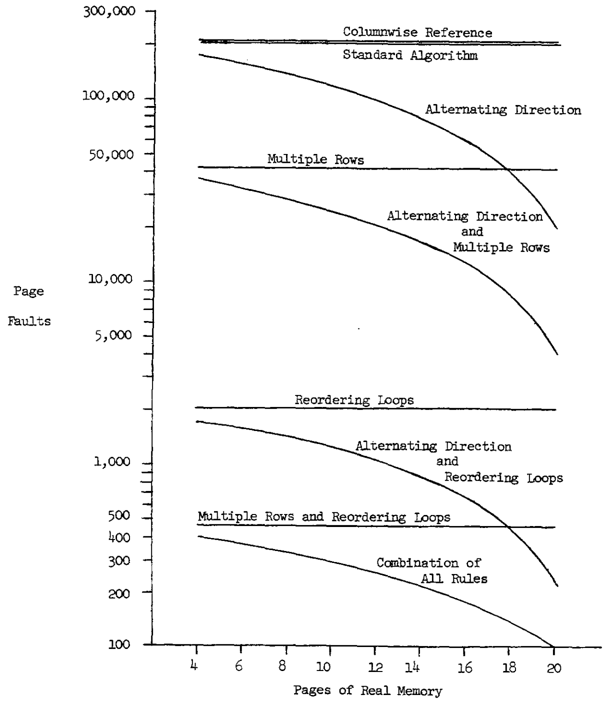


Figure 4—Comparison of matrix multiplication algorithms for matrices spanning twenty pages

and executed. The environment was like that described for the transpose operation in Section IV. Tables V and VI show the measurements made. For those algorithms generating more than 2500 page faults, partial runs were completed and the results extrapolated.

Table VI illustrates several significant points about the application of the programming rules to matrix multiplication. The increase in problem CPU time is about the same for each rule or combination of rules. The elapsed time to complete the specified matrix multiply was reduced from 5.4 hours to 4.2 minutes by applying the rules. The programmer should consider all three rules. After applying the reordering rule, the programmer could be satisfied with a 99.9 percent decrease in page faults in exchange for a 12.7 percent increase in problem CPU time. By continuing with the other two rules, the problem CPU time is increased an additional 0.2 percent while the page faults are reduced by another 88.4 percent.

The matrix multiplication program which resulted when all three rules were applied to the standard program and then used to generate the data shown in Tables V and VI is shown as Example 2.

```
INSTART = 1;
INEND = M;
INDIFF = 1;
DO BASEROW = 1 TO L BY RPP;
   LASTROW = MAX(BASEROW + RRP-1, L);
   DO ROW = BASEROW TO LASTROW BY 1;
      DO COL = 1 TO N BY 1;
         C(ROW,COL) = 0;
      END;
   END;
   DO INNER = INSTART TO INEND BY
     INDIFF;
      DO ROW = BASEROW TO LASTROW
        BY 1;
         DO COL = 1 TO N BY 1;
            C(ROW,COL) = C(ROW,COL) +
              A(ROW,INNER) * B(INNER,
              COL);
         END;
      END;
   END;
   TEMP = INSTART;
   INSTART = INEND;
   INEND = TEMP;
   INDIFF = -INDIFF;
END;
```

Example 2—Matrix Multiplication With All Rules Applied

REMARKS ON APPLICABILITY AND SUMMARY

In this paper the problem of performing matrix operations on large matrices is being considered from the point of view of the application programmer. The motivation is to reduce the number of page faults encountered while performing the

TABLE V—Summary of Matrix Multiplication Data

Rule			Expected	Measured	Measured
1	2	3	Page Faults	Page Faults	Expected
			204060	204266	1.001
X			2060	2098	1.018
	X		42460	42629	1.003
		X	71460	71986	1.007
X	X		460	499	1.084
X		X	760	873	1.148
	X	X	14900	16306	1.094
X	X	X	210	314	1.495

TABLE VI—Matrix Multiply Resource Utilization

Rule			Problem	System	Total	Elapsed	I/O
1	2	3	CPU	CPU	CPU	Time	Time
			197.3	4493.9	4691.2	19460	14768.4
X			222.3	46.2	268.5	420	151.7
	X		221.7	937.8	1159.6	4242	3082.1
		X	224.5	1583.7	1808.2	7013	5204.6
X	X		222.6	11.0	233.6	269	36.1
X		X	222.7	19.2	241.9	305	63.1
	X	X	229.6	358.7	588.3	1767	1178.9
X	X	X	222.7	6.9	229.6	252	22.7

Units are seconds.

matrix operation in order to improve the performance of the application program. Three rules have been given which may be applied by the application programmer to a source program in order to reduce the number of page faults. (1) Nest loops so that matrix elements are referenced in the same order as they are stored. (2) Process all the rows in one page while the page is in real memory. (3) Alternate the direction of traversing a matrix to reuse pages not purged from real memory.

When virtual memory was first introduced, one of its major advantages was said to be that of allowing the programmer to work in a real memory environment without concern about overlays. Shortly thereafter, material began to appear discussing the locality and compactness of a program. Some papers actually discussed program design in terms of the average number of real memory pages an operating system would allocate to the application. Clearly, the freedom of the programmer is abridged when space considerations must be made.

Applying the rules described in this paper does not really diminish the freedom of the programmer but does allow the programmer to get better performance from the application program by using additional knowledge. The programmer may use knowledge of (1) the matrix mapping function of the language, (2) the word size and page size of the machine, (3) the paging algorithm, or (4) some combination of these items in order to reduce the number of page faults generated by the program.

The data presented show that using the rules will not only improve the performance of the application but may also greatly lessen the demands on the resources of the system. For a slight increase in problem CPU time, reductions can be realized in total CPU time, elapsed time or real memory costs, and channel time. Yet, when both a FORTRAN library and a PL/I library on a paging system were checked, the standard matrix transpose operation and the standard matrix multiplication operation used as examples in this paper were programmed.[10,11] Several other matrix operations that were checked could have been easily improved in an obvious manner. In addition to program libraries, individuals concerned with program performance should be aware of the code executed when a program refers to all of the elements of matrix by a simple reference to the matrix by name. PL/I has several of these matrix operations defined. Also, several languages allow the notation A(I, *) to refer to all of the elements in row I of matrix A. Program performance may be improved by explicitly writing the loop controls to access all of the elements in the row.

In an early part of this paper, the class of problems was restricted to those in which at least one row of a matrix could be contained in a page. That restriction was for the purpose of deriving formulas only. The programming techniques apply to matrices of any size. In fact, the programming rules may be applied to the problem of folding in processing large matrices in a non-paging environment.

The importance of applying a rule so that the number of page faults depends on the amount of real memory available should not be overlooked. For example, only the alternating direction rule introduced this dependency into the matrix multiplication operation. All of the matrix multiplication algorithms not employing alternating directions would have performed the same in three pages of real memory as in any larger number of pages until the point at which all of the data would fit in real memory. On the other hand, those algorithms, which depend on the amount of real memory available, had better performance for each page of real memory allocated to them.

BIBLIOGRAPHY

1. Sayre, D., Is Automatic "Folding" of Programs Efficient Enough to Displace Manual?," *CACM 12*, 12 (December 1969), pp. 656-660.
2. Denning, P. J., "The Working Set Model for Program Behavior," *CACM*, May 1968, pp. 323-333.

3. Belady, L. A., "A Study of Replacement Algorithms for a Virtual Storage Computer," *IBM Syst. J.* 5, 2, 1966.
4. Coffman, E. G. and L. C. Varian, "Further Experimental Data on the Behavior of Programs in a Paging Environment," *CACM 11*, 7 July 1968, pp. 471-474.
5. Brawn, B. S., F. G. Gustavson, and E. S. Mankin, "Sorting in a Paging Environment," *CACM 13*, 8, August 1970, pp. 483-494.
6. Moler, C. B., "Matrix Computations with Fortran and Paging," *CACM 15*, 4, April 1972, pp. 268-270.
7. Dubrulle, A. A., "Solution of the Complete Symmetric Eigenproblem in a Virtual Memory Environment," *IBM J. Res. Dev.* 16, 6, November 1972, pp. 612-615.
8. McKellar, A. C. and E. G. Coffman, "Organizing Matrices and Matrix Operations for Paged Memory Systems," *CACM 12*, 3 March 1969, pp. 153-165.
9. Guertin, R. L., "Programming in a Paging Environment, *Datamation*, February 1972, pp. 48-55.
10. —— *System/360 Scientific Subroutine Package*, Version III, Programmer's Manual, IBM Edition GH20-0205-4, August 1970, White Plains, New York.
11. —— *System/360 Scientific Subroutine Package (PL/I)*, Program Description and Operations Manual, IBM Edition H20-0586-0, January 1968, White Plains, New York.

The double paging anomaly

by ROBERT P. GOLDBERG
Harvard University
Cambridge, Massachusetts

and

Honeywell Information Systems
Waltham, Massachusetts

and

ROBERT HASSINGER
Liberty Mutual Insurance Company
Hopkinton, Massachusetts

INTRODUCTION

Belady's paging anomaly[1] has illustrated that certain page replacement algorithms can cause more page faults as the size of memory increases. Mattson[2] has shown that there exists a class of algorithms called stack algorithms (such as LRU least recently used) which cannot cause more page faults as memory size increases. In this paper, we investigate the dynamics of double-paging, i.e., running a paged operating system, e.g., IBM's OS/VS2,[3] under a paged virtual machine monitor, e.g., VM/370.[4] In particular, we show that an increase in the size of the memory of the virtual machine without a corresponding increase in its real memory size can lead to a significant increase in the amount of paging, even for the LRU algorithm.

DOUBLE PAGING

Virtual machine systems[4–8] provide the environment in which double paging phenomena can occur.* The core of a virtual machine system is the virtual machine monitor or VMM. The VMM has certain similarities to conventional operating systems in that it supports a user interface or "extended machine" on which user programs can be run. However, the extended machine supported by a VMM is the full functional counterpart of an existing computer, and may be the same computer that the VMM is itself running on. Since the VMM can support programs which utilize the full functionality of a computer system, the VMM can support complete operating systems in the same way that operating systems support user programs. It is thus possible to run two incompatible operating systems on a single computer at the same time, or to run one operating system in production mode while system programmers are simultaneously modifying another active copy of that same operating system.

Figure 1 uses IBM's VM/370 to illustrate the organization of a virtual machine svstem. The VM/370 Control Program (the VMM) is shown, running on a bare System/370 and supporting two virtual machines, VM1 and VM2. Since the virtual machines are identically equivalent to complete System/370 computer systems, any of the System/370 operating systems, such as OS/VS2 can be run on virtual machine VM1. OS/VS2, in turn, supports an "OS/VS2 Extended Machine" on which a user program is being run.

While paging is not an essential requirement for a virtual machine system,[5,7,9] the two facilities form a very powerful combination together. In particular, with paging, it becomes possible for the memory of the virtual machine(s) created to be larger than the real memorv. This facility has been used very effectively by CP-67 and VM/370 to run various operating systems and applications which require a very large memory.[4,10,11]

If the VMM supports a virtual machine which includes paging, then it is possible to run any paged operating system (including the VMM) on this virtual machine. In this case, we define:

- Level 2 memory—virtual memory of virtual machine
- Level 1 memory—memory of virtual machine
- Level 0 memory—memory of real machine, i.e., real memory.

The Level 2 memory is mapped via paging into Level 1 memory. The Level 1 memory is mapped, in turn, into Level 0 memory.* Under these circumstances. we have *double paging.*

Conventional computer systems do not provide direct

* In this paper we use the term "double paging" to refer only to these phenomena. It does not refer to any other popular uses of this term.

* The term *level* is used here informally rather than in the strict sense as defined in Goldberg.[6,7]

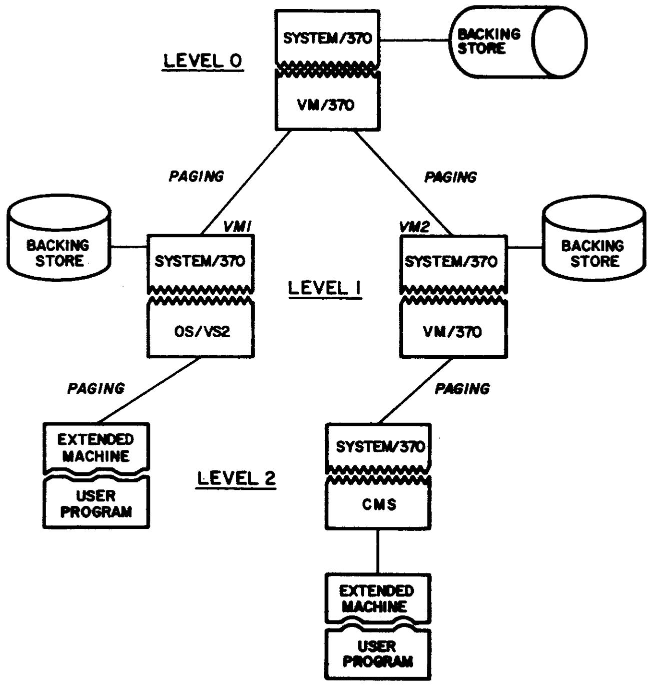


Figure 1—VM/370 virtual machine organization illustrating double paging

hardware support for double paging. Thus, in systems such as VM/370, software must be used to simulate the double paging *mechanism.* When a process is to be activated in Level 2 memory, its page mappings from Level 2 to Level 1 and from Level 1 to Level 0 must be combined to yield a single composed mapping directly from Level 2 to Level 0. Goldberg[7,12] and Parmelee[11] discuss the *mechanisms* for software support of double paging in conventional computer systems and Goldberg[6,7] discusses proposed computer architectures, called *virtualizable architectures* which typically provide direct hardware support for efficient double paging mechanisms. In this paper we shall not examine mechanisms. Rather we shall be concerned only with *policies*, i.e., the page replacement algorithms, and see how thev are affected by double paging.

Figure 1 illustrates two examples of double paging which can arise in the operation of VM/370. VM/370 uses paging to create the (illusion of) memory for virtual machines VM1 and VM2. OS/VS2 which is running on VM1, in turn, uses paging to create a large address space (virtual storage) extended machine for its user programs. On VM2, another copy of VM/370 is running, producing a second level of virtual machines. In the figure, the second level virtual machine is running CMS, an operating system which does not utilize the paging mechanism. Thus, both user programs shown in the figure will be affected by double paging.

Every paged system requires a backing store to preserve each page's contents when the page is not in the real or virtual memory. In Figure 1, the backing store shown at Level 0 holds pages of Level 1 (VM1 or VM2) which are not resident in Level 0 memory. The two Level 1 backing stores, in turn, hold pages of the Level 2 memory which are not resident in Level 1 memory.

The illustration of Figure 1 does occur in real world situations. As paging becomes more common in "target" machines, double paging will become more common in encapsulated systems. Examples of encapsulated systems include both the virtual machine systems (as illustrated above) for identical host and virtual machines, and integrated emulators[13] for dissimilar (paged) machines. An example of the latter might be running the PDP-10 TENEX System under an integrated emulator under OS/VS2*.

Other environments where double paging can be expected are in the newly proposed complex virtualizable architectures.[6,7,14] Related considerations arise in the management of multi-level (three or more) memory systems.[15]

DYNAMICS OF DOUBLE PAGING

We will examine the effect of choice of page replacement algorithms and sizes of memory upon the dynamics of double paging. We restrict our attention to demand algorithms operating in fixed memory spaces. Thus we will study one virtual machine at a time and ignore other effects introduced by resource multiplexing among VMs. Furthermore, when we examine the paging behavior of a reference string, we examine the behavior of the original string. We ignore any effective "rewriting" (or renaming) of the string which might occur.

Thus:

(1) We ignore "interference" to the reference string caused by any VMM traps and simulation.
(2) We assume that pages are treated homogeneously by algorithms, i.e., pages are not "locked."
(3) We assume page replacement algorithms and tables are not themselves in virtual memories. We assume they are external to the system or in hardware.

Loosening these ground rules makes the analysis more complex and might become the basis for future studies.

Furthermore, we make the following assumptions:

(1) Memory sizes in number of page frames are called n_0, n_1, n_2.
(2) n_0, n_2 with $n_2 \geq n_0$ will be known and fixed.
(3) We can set n_1 but once set it will be fixed.
(4) The same algorithm is used for level $2 \rightarrow 1$ and level $1 \rightarrow 0$ paging.
(5) A demand page replacement algorithm is used and all free pages will be utilized, i.e., $n_1 \geq n_0$, $n_2 \geq n_1$.
(6) We count total page faults and ignore the fact that different backing store devices might be used at each level.

With the above assumptions and terminology we can

* This is merely a hypothetical example.

identify four distinct operating regions. They are illustrated in Figure 2.

(a) $n_0 = n_1 = n_2$
This case arises if $n_0 = n_2$. Then by assumption 5 (above) n_1 must equal n_0 and n_2. In this case, after the initial pages are brought into each memory no additional paging occurs.

(b) $n_0 = n_1 < n_2$
This case arises if $n_0 < n_2$ and we choose to set $n_1 = n_0$. In this case, after the initial pages are brought into level 0 memory there is only paging for level 2→1.

(c) $n_0 < n_1 = n_2$
This case arises if $n_0 < n_2$ and we choose to set $n_1 = n_2$. In this case, after the initial pages are brought into level 1 memory there is only paging for level 1→0.

(d) $n_0 < n_1 < n_2$
This case arises if $n_0 < n_2 - 1$ and we choose some intermediate value for n_1. Paging activity occurs for level 2→1 and level 1→0.

Case (a) is trivial and uninteresting. After start up, cases (b) and (c) exhibit identical behavior to a one level paging system. However, these cases remain of interest for comparison with the double paging case (d).

LRU ALGORITHM APPLIED TO DOUBLE PAGING

As noted above, stack algorithms such as LRU (least recently used) have a number of desirable properties. Among

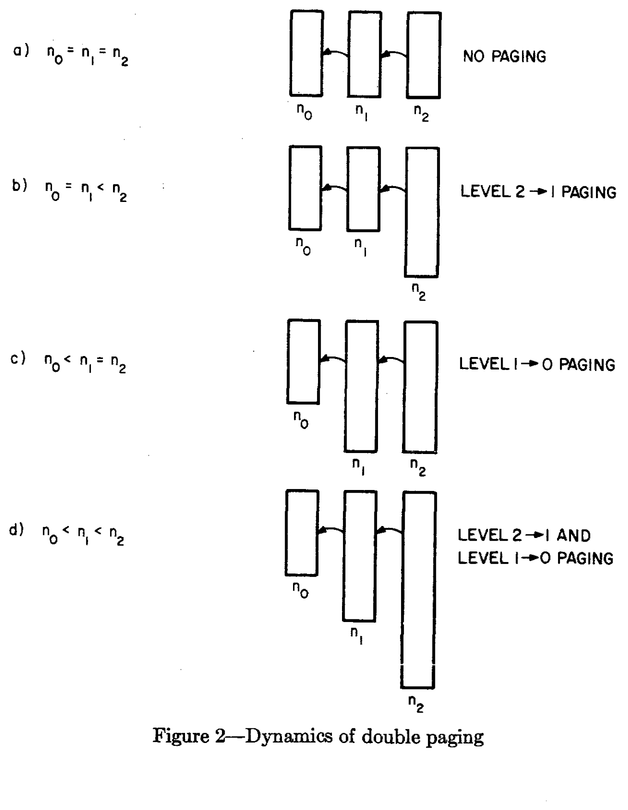


Figure 2—Dynamics of double paging

LRU 2 pages $n_0 = 2$ $n_1 = 5$	1	2	3	2	4	2	5	2	1	4	1	5	Reference String	
	*	*	*		*		*		*	*		*	Faults	4 Faults
	1	2	3	2	4	2	5	2	1	4	1	5	Stack	
		1	2	3	2	4	2	5	2	1	4	1		
LRU 3 pages $n_0 = 3$ $n_1 = 5$	*	*	*		*		*		*	*		*	Faults	
	1	2	3	2	4	2	5	2	1	4	1	5	Stack	4 Faults
		1	2	3	2	4	2	5	2	1	4	1		
			1	1	3	3	4	4	5	2	2	4		
LRU 4 pages $n_0 = 4$ $n_1 = 5$	*	*	*		*		*		*				Faults	
	1	2	3	2	4	2	5	2	1	4	1	5		
		1	2	3	2	4	2	5	2	1	4	1	Stack	2 Faults
			1	1	3	3	4	4	5	2	2	4		
					1	1	3	3	4	5	5	2		

Figure 3—Single level LRU: Increase in memory size cannot increase number of faults

these is the stack property that any increase in memory size cannot cause an increase in the number of page faults.[2]

In Figure 3, we illustrate the stack property by applying the LRU algorithm to a reference string which is run in a (conventional single paging) memory with three different sizes. The figure indicates page faults with an asterisk (*), and shows the stack contents at each point in time. In order to provide a uniform basis of comparison for all examples, faults are counted only after the largest stack has been filled.

The largest stack size is 4 and this stack will be filled after reference string elements 12324 have been run. We delimit this starting point with a dashed vertical line. and in Figures 3-6 count only those asterisks which fall to the right of the dashed vertical line. When the reference string consisting of five distinct page names ($n_1 = 5$) is run in a two page memory ($n_0 = 2$), four faults occur. When memory is increased to three pages ($n_0 = 3$), four faults still occur. When memory is increased to four pages ($n_0 = 4$), the number of faults drops to two.

In double paging, we must specify the reference string (same as above), the replacement algorithm (LRU), and the memory sizes (to be indicated below). Figures 4-6 apply LRU algorithms at both levels to the same reference string. We fix the reference string memory $n_2 = 5$ and the real memory $n_0 = 2$. Then, we count the number of faults as the virtual machine memory n_1 is varied.

Figure 4-6 are similar to the single level paging case of Figure 3. The reference string, stack values, faults, and beginning of count (dashed vertical line) are shown. However, in these figures, the TOTAL faults are given by the sum of level 2→1 faults and level 1→0 faults.

In Figure 4, reference string memory $n_2 = 5$, virtual machine memory $n_1 = 2$, and real memory $n_0 = 2$. After start-up we count 4 Level 2→1 faults and 0 Level 1→0 faults for a total of 4 faults. This is double paging case (b).

Figure 5 illustrates the anomalous double paging behavior which can arise in case (d). We keep the same real memory size $n_0 = 2$, and the same reference string memory $n_2 = 5$.

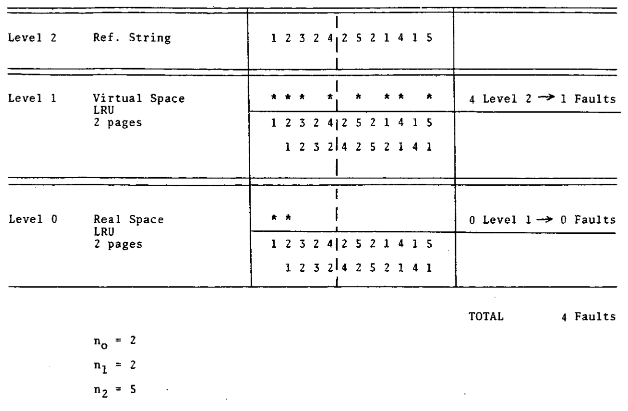


Figure 4—Double paging with LRU: Case (b)

However, we increase the size of the virtual machine's memory $n_1=3$. As can be seen, the paging behavior becomes significantly worse. There are still 4 Level 2→1 faults but now there are also 4 Level 1→0 faults for a total of 8 faults. Thus, the number of faults has doubled. Figure 5 details how this increase has occurred. The small circles in Figure 5 indicate those pages in Level 1 which cause an extra induced fault in Level 0. The arrows point from the Level 1 pages to the extra Level 0 pages which are needed. Finally, the large circles in Level 0 indicate page renaming operations, i.e. what used to be page 1 is now page 4. For the LRU algorithm, each Level 2→1 fault causes an extra Level 1→0 fault to occur also.

In Figure 6, we further increase virtual machine memory size $n_1=4$, keeping reference string memory $n_2=5$ and real memory $n_0=2$. There are still 4 Level 1→0 faults but the number of Level 2→1 faults has dropped to 2. Thus, the page fault total has decreased to 6.

We can summarize the results of this example as:

For a given reference string in a double paging system, an increase in the size of the memory of the virtual machine without a corresponding increase in its real memory size can lead to a significant increase in the number of page faults, even for the LRU algorithm.

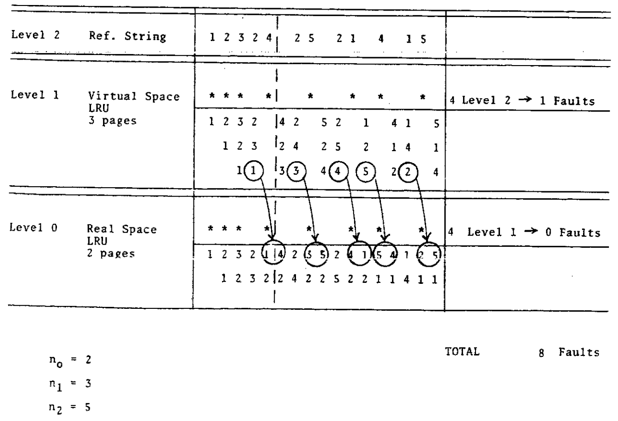


Figure 5—Double paging with LRU: Case (d)

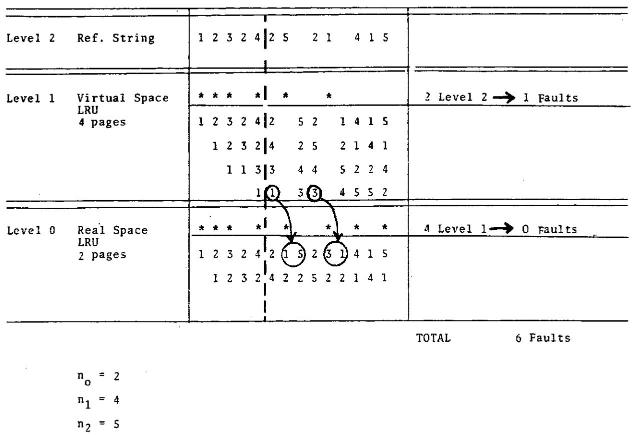


Figure 6—Double paging with LRU: Case (d)

THE ANOMALY EXPLAINED

The double paging anomaly occurs when $n_2>n_0+1$, n_0 and n_2 are fixed and we have $n_1=n_0$ (Case (b)). We increase $n_1=n_0+1$ (Case (d)) and the amount of paging increases significantly.

The anomaly can be explained for the LRU algorithm by an inspection of the reference stacks of Figure 5. The level 1 stack contains three entries whereas the Level 0 stack contains only two entries. When a page is at the bottom of the level 1 stack and is a candidate for next removal, it already has fallen out of the level 0 stack and has been swapped. Thus, in order to swap an entry from the level 1 stack, it must first be swapped into the level 0 stack.* Since LRU algorithms are operating at both levels, the level 0 removal algorithm will make exactly the worst choice each time.

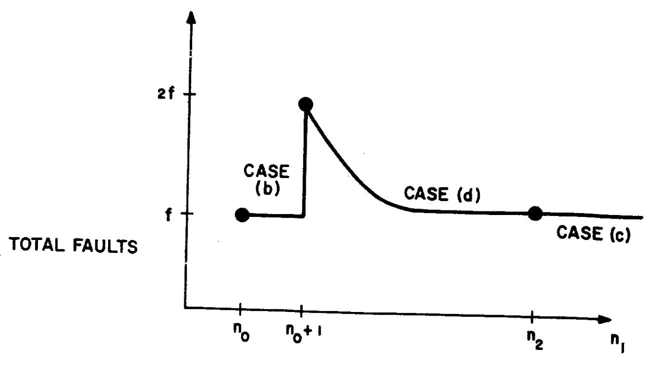


Figure 7—Page fault dependence on n_1 for LRU algorithm

* When an entry is "swapped" into a stack, its corresponding page is swapped into memory.

Thus, it will always have to swap in the page which it most recently swapped out.

For the LRU algorithm, worst performance occurs for Case (d) with $n_1=n_0+1$. As n_1 increases, the number of page faults continues to decrease (the stack property) until $n_1=n_2$ and we have Case (c). While the actual performance depends upon the reference string and memory sizes, the trend can be seen in Figure 7. The figure is drawn as a continuous curve even though it is really a series of steps.

Surprising results hold not only for the LRU algorithm but for other double paging replacement algorithms as well. For example, with the same reference string used above, the FIFO (first in first out) algorithm also doubles from 5 to 10 the number of page faults in going from $n_1=2$ to $n_1=3$. On the other hand, with the very unlikely MRU (most recently used) algorithm, the number of page faults remains constant as n_1 is varied between $n_1=2$ and $n_1=5$.

VM/370 avoids some of the difficulties explored in this paper through the use of certain specialized algorithms which allow locking (dedicated) pages in main memory. While this procedure may decrease susceptibility to the double paging anomaly, it reduces resources available to other users and might adversely affect global performance. In any case, we have shown that in double paging situations great care must be exercised.

As noted above, virtual machine recursion[5,7] implies the ability to run a VMM under a VMM under a VMM . . . . It is known that in order to test VM/370 software before System/370 hardware was available, IBM ran specially modified versions of CP-67 several levels deep. In the new complex virtualizable architectures *mechanisms* are provided for supporting arbitrarily deep recursion. In these systems, the double paging problem generalizes to the m-level paging problem.

CONCLUSION

Progress in understanding complex phenomena is often made through the discovery and explanation of anomalous behavior which arises in apparently simple situations. In this paper we have examined one aspect of the resource allocation problem in a large computer system. We have observed that when rational locally optimal algorithms are combined together, distinctly suboptimal global behavior can sometimes result.

REFERENCES

1. Belady, V. A., "A Study of Replacement Algorithms for a Virtual-Storage Computer," *IBM Systems Journal*, Vol. 5, No. 2, 1966.
2. Mattson, R. L., J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Systems Journal*, Vol. 9, No. 2, 1970.
3. IBM, *Introduction to OS/VS2 Release 2*, IBM Corporation Publication No. GC28-0061.
4. *IBM Virtual Machine Facility/370—Planning Guide*, IBM Corporation, Publication No. GC20-1801-0, 1972.
5. Buzen, J. P., U. O. Gagliardi, "The Evolution of Virtual Machine Architecture," *Proceedings AFIPS National Computer Conference*, 1973.
6. Goldberg, R. P., "Architecture of Virtual Machines," *Proceedings AFIPS National Computer Conference*, 1973.
7. Goldberg, R. P., *Architectural Principles for Virtual Computer Systems*, Ph.D. Thesis, Division of Engineering and Applied Physics, Harvard University, Cambridge, Massachusetts, 1972.
8. Goldberg, R. P. (ed.), *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
9. Goldberg, R. P., "Virtual Machines: Semantics and Examples," *Proceedings IEEE Computer Society Conference*, Boston, Massachusetts, 1971.
10. Meyer, R. A. and L. H. Seawright, "A Virtual Machine Time Sharing System," *IBM Systems Journal*, Vol. 9, No. 3, 1970.
11. Parmelee, R. P., T. I. Peterson, C. C. Tillman and D. J. Hatfield, "Virtual Storage and Virtual Machine Concepts," *IBM Systems Journal*, Vol. 11, No. 2, 1972.
12. Goldberg, R. P., *Virtual Machine Systems*, MIT Lincoln Laboratory Report No. MS-2687, (also 28L-0036), Lexington Massachusetts, 1969.
13. Mallach, E. G., "Emulation—A Survey", *Honeywell Computer Journal*, Vol. 6, No. 4, 1973.
14. Lauer, H. C. and D. Wyeth, "A Recursive Virtual Machine Architecture," *Proceedings ACM SIGARCH-SIGOPS Workshop on Virtual Computer Systems*, Cambridge, Massachusetts, 1973.
15. Scheffler, L. J., "Optimal Folding of a Paging Drum in a Three Level Memory System," *Proceedings of ACM SIGOPS Fourth Symposium on Operating Systems Principles*, Yorktown Heights, New York, 1973.

Effective planning for and justification of the extension of data processing in hospitals

by RICHARD B. FREIBRUN

Compucare, Inc.
Chicago, Illinois

Starting with the basic premise that: There is a significant role for data processing as a viable tool to assist in patient care and administrative management of hospitals, then we can dispense with the assumption that it should no longer be necessary to convince administrators of the need for computerization in hospitals. This premise seems to be substantiated by the significant increase in the application of computer technology over the last several years as documented by a 1972 American Hospital Association Survey that indicated that of 552 hospitals sampled, 81 percent had one or more in-house computers and an additional 5 percent used out-of-house computer services.

By virtue of today's socio-economic environment and continuing advancements in medicine it is a given fact that hospitals are becoming more complex and are offering more services. The management of a more complex and diverse institution offering a broader range of services becomes much more difficult. To compound the complexity problem, there continues to be a shortage of qualified professional personnel and those qualified professionals now in the field are being used heavily in clerical tasks. All of these trends make the hospital a very difficult institution to manage.

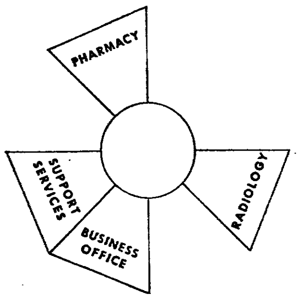
In addition to the pressures on the industry caused by complexity, there have also been a number of clearly visible trends related to attempts to solve this problem through the use of data processing technology. Within the last year there have been many clearly recognizable trends.

- A greater number of vendors with installed systems.
- A number of new vendors entering the field.
- A number of established vendors leaving the field.
- Heightened interest and understanding of integrated clinical data processing concepts by administrators.
- Pressures for internal cost reduction and quality of care justifications caused by Phase III, Phase IV, C.O.L.C., P.S.R.O.'s, etc.
- The need for new management information to insure C.O.L.C. "guideline" compliance.
- Continued growth of professional societies in the Health Care Data Processing area, i.e., HISSG (Hospital Information Systems Sharing Group), Society for Computer Medicine, HMSS (Hospital Management Systems Society).
- AHA Interest: Advisory Panels
  Numerous Institutes
- The recognition by administrators that Data Processing is an expensive resource that must be managed as such.
- The growing recognition of the need to justify programs of computerization before investments are made with the corresponding subsequent requirement of "tracking" the performance of that investment once made.
- The recognition by administrators that the use of Data Processing in hospitals has been costly with marginal return, if any, from the computer investment.

Currently, computers in most hospitals are employed in financial applications, such as patient billing, payroll and accounts receivable. The pressures of increased workloads and insurance company and other agency reporting requirements make the automation of these functions often necessary and certainly useful. However, this rarely yields an appropriate return on a data processing investment, except occasionally in terms of improved cash flow through the more efficient control and more rapid collection of accounts receivable.

The real benefit of automation lies in its use in the handling of information in the clinical departments. The large quantity of information pertaining to the care of the patient which is processed in these areas provides excellent justification for properly executed programs of extended automation. Certain clinical departments lend themselves more readily to automation because of their high volume of data and their high cost of operation, typically represented by personnel costs, i.e., Laboratory, Radiology, Central Supply, Pharmacy, Dietary. Nursing, representing approximately 40 percent-50 percent of a hospital's budget, becomes an excellent source for systems improvement opportunities through the reduction of clerically intensive tasks performed as a result of a physician's order for a clinical service.

A relatively small number of hospitals have proceeded with programs of extended automation beyond the Business Office. These systems typically have involved one or more



LOWER G.I. SERIES
(FOR TOMORROW)

	STEPS	DOCUMENTS	INDEPENDENT FILES
NURSING UNIT	30	12	7
RADIOLOGY	18	6	5
SUPPORT SERVICES			
- DIETARY	5	4	1
- TRANSPORTATION	8	2	1
PHARMACY	12	4	3
BUSINESS OFFICE	9	4	2
TOTAL	82	32	19

Figure 1

of the following features:

- The collection of the doctor's order at the source (the nursing unit).
- The transmission of that order to the interested service department (Lab, X-Ray, Pharmacy, etc.).
- The processing of the information (patient charge posting, Pharmacy inventory update, Lab test result).
- The retrieval of the information (transmission and display at the source for use in the care of the patient).
- The ultimate storage of the information as a component of the patient's medical record.

The experience of hospitals in the past, which have ventured beyond the Business Office, has indicated that many seemingly viable automation programs have not produced the results at the costs anticipated for them. Difficulties have been encountered in these programs primarily for the following reasons:

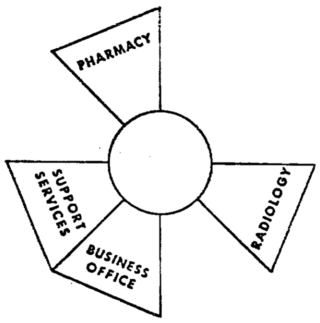
- The selected approach to automation has required large front-end investments in anticipation of future benefits (cost reduction, better quality of care, etc.). This high risk approach, accompanied by the failure to achieve the desired benefits, has resulted in a waste of resources and tremendous dissatisfaction on the part of the hospitals involved.
- The selected approach to automation did not fit the hospital's or the vendor's ability to achieve. When programs were undertaken, well thought out plans for implementation were not developed, and activities were not carefully monitored—as a result, the benefits were not realized.

Although benefits, as a rule have not been achieved, a well planned program can indeed:

Reduce Costs
Increase Revenue
Enhance Quality of Care (Through Better Accuracy and Timeliness)
Free Professional Personnel from Clerical Tasks
and
Reduce Systems Complexity and Opportunities for Error

This may appear as an impossible set of goals but when the complexity of information processing within a hospital is better understood, the goal is more achievable.

Illustrative of this point is the ordering of a lower G.I. series procedure. This is a common procedure which typically affects a number of areas in the hospital. At one hospital which we have examined in depth, we found that the manual system they were using to communicate information required 82 steps, 32 separate documents and resulted in the filing of 19 documents (Figure 1). The Nursing Unit was involved in both sides of the procedure, Radiology, of course, took the pictures, Dietary was involved in a special diet for the patient, Transportation was involved to bring the patient to Radiology and back to the Nursing Unit, Pharmacy was involved in providing certain preparatory drugs, the Business Office, of course, was involved in the billing. Through a proposed automation technique at this particular hospital, the steps involved were reduced from 82 to 23, the documents involved from 32 to 5 and the documents filed from 19 to 2



LOWER G.I. SERIES
(FOR TOMORROW)

	STEPS	DOCUMENTS	INDEPENDENT FILES
NURSING UNIT	~~30~~ 8	~~12~~ 1	~~7~~ 1
RADIOLOGY	~~18~~ 7	~~6~~ 3	~~5~~ 1
SUPPORT SERVICES			
- DIETARY	~~5~~ 0	~~4~~ 0	~~1~~ 0
- TRANSPORTATION	~~8~~ 4	~~2~~ 0	~~1~~ 0
PHARMACY	~~12~~ 4	~~4~~ 1	~~3~~ 0
BUSINESS OFFICE	~~9~~ 0	~~4~~ 0	~~2~~ 0
TOTAL	~~82~~ 23	~~32~~ 5	~~19~~ 2

Figure 2

(Figure 2). By decreasing the number of steps, cost reduction could be obtained and the quality of care of that particular hospital could be enhanced.

One of the major factors impacting "quality" of care is "opportunity" for error (Figure 3). It follows that the more tasks that must take place, the more opportunity for error. By reducing the tasks that take place manually, the opportunity for error is correspondingly reduced. It has been our experience that those innovations which reduce costs in a hospital are the same things that enhance the quality of care.

As a result of internal and external pressures and obvious opportunities for improving hospital operations, many hospitals have approved large programs for the extended use of the computer. Most hospitals have not, however, achieved the expected benefits intuitively projected for these programs. In actuality, automation has increased costs, had little affect on patient care, increased the complexity of managing the hospital and increased the management burden. Why is this true? The prime causes of lack of complete success in computer effort results from a number of factors:

There has been an oversimplification of the problem coupled with a general lack of recognition that you can't change overnight. You must change in a pre-planned careful fashion and the "Management of Change" requires skills above and beyond those that might be currently available within the Hospital Organization. Too frequently, hospitals have acted intuitively in this matter. They feel that the computer will be beneficial but they do not define the benefits and the plan to realize those benefits. Instead they typically move ahead without clear justification or clear computer program objectives. When the computer program does not achieve what administration expected, they are, of course, unhappy. This might be characterized as a function of inadequate planning.

Hospital computer programs being considered today typically require large front-end investments and thus automatically become high risk programs particularly when measured in terms of achieved benefits. The large commitment necessary at the front end will lock a hospital into a program from which there is no turning back. Thus with inadequate planning, the program becomes a very high risk affair.

Another factor is the over excessive influence by vendors and suppliers in the decision making process. Vendors, by their nature, tend to oversell and encourage large commitments. Unless the hospital grabs hold of the problem and adequately defines its needs in a preplanned way, it risks too much vendor control and influence of the program.

The proliferation of vendors in the field causes still another difficulty. Each vendor has his own approach, plan and product and makes a number of claims which are appealing while attempting to establish his difference. Because of the large number of vendors, it is extremely difficult to evaluate capabilities and make an intelligent choice of a program.

In examining some of the general characteristics of hospitals considering extending data processing into so called "HIS/MIS" programs, we find that they are typically larger and therefore more complicated hospitals who can potentially gain a greater benefit from computerization and have certain characteristics in common:

OPPORTUNITIES FOR ERROR

	Common Pieces of Information	Number of Documents	Number of Transcriptions	Opportunities for Error
Nursing Unit	15	5	5	375
Pharmacy	6	4	3	72
Business Office	10	4	1	40
			TOTAL	487

Figure 3

- These hospitals almost universally are concerned about the problem of using the computer more extensively and rank the computer problem high amongst those that need to be solved in the next several years.
- They are very acutely aware of previous failures and are very concerned about making a major commitment in light of the less than successful results of the industry.
- They are presently spending a lot of money on computers.
- These hospitals are generally spending between two and four dollars per patient day currently, and are using the computer primarily in the Business Office, with varying degrees of effectiveness.

The thing that they all have in common is that they recognize the need for even further expenditures and are presently trying to decide upon their next step while considering questions like:

- Should we use the computer outside the Business Office or shouldn't we?
- Should we consider a shared service vendor instead of our in-house computer?
- Should we make a full-fledged commitment to a "total" HIS/MIS?
- Should we buy a package?
- Should we develop our own?
- What application first? Why?
- How much will each application cost? Why?
- How estimated?
- What are the varying costs, benefits and risks of the different next steps?
- What effect do regulatory and community agencies have?

The problem is very complicated. There are a number of conflicting pressures which have resulted in an ambivalent attitude and a lack of decisiveness toward the extended use of computers in hospitals. On the one hand, the administrator knows that he will have to use the computer more extensively but on the other hand, he is unhappy at the prospect of greater expenditures in an area whose performance has been mixed, at best. The problem is more of *what to do* rather than *what not to do.*

In addition to the difficulty caused by these factors there is a general lack of recognition of the requirements necessary for managing an automation effort. There is a tendency to use inadequate control methods and to inaccurately define the goals of the program. This has led to underestimates of costs, and overestimates of benefits. If there is anything that makes management unhappy, it's underestimates of costs and overestimates of benefits. This has been followed by a communication gap between management and computer technologists. There has generally been too much emphasis on technology and too little emphasis on establishing:

- Proper program management
- Proper program objectives
- Proper program planning
- Proper system design
- Proper project control and tracking

It is in this area that hospital management falls down.

The technology required to make "hospital information systems" work has been available for several years and has been appropriately used in a number of industries. The complexity of the hospital, however, makes it more difficult to use computers in that environment than in most other environments. This is not a technical problem. This is a management problem and should be treated from that perspective.

In approaching this problem there are a number of questions which must be asked and answered before a satisfactory solution can be achieved.

- What should be done first?
- What should be done second?
- What should be done third?
- Which applications are feasible?
- Which applications are justifiable?
- In what order should you prioritize your program?
- How much should be spent? Over what period of time?
- How will such expenditures be justified?
- Who should I expend with?
- Over what length of time can the ultimate system be reasonably installed? Is it six months, one year, ten years, fifteen years?
- How can I be reasonably certain of the accuracy of my plan?
- What are the checkpoints which tell me I'm succeeding—and should proceed—or failing—and should stop and re-evaluate?

All of these questions must be carefully addressed before undertaking implementation of a program. A proper approach to the extended use of computers in hospitals must include the use of sound business justification and planning techniques preliminary to the start of implementation of the program.

Compucare, through its experience with many hospital automation projects, has developed techniques which support hospitals in these difficult decisions and which tend to greatly reduce the risk of their actions. Utilizing the minimum criteria of:

- Reduction to cost
- Increase to revenue and
- Enhancement to patient care

as key indicators against which to proceed. The approach we have developed and apply generally follows this guideline:

1. Assess the present computer program to determine the effectiveness and benefits of the current Data Processing budget.
2. As a result of the assessment a plan of action should be developed to improve the existing program in the short range considering:

   —Computer capacity—over/under
   —Projects to "kill"
   —Staff and development techniques
   —Available budget dollars

3. The definition of the benefits of automation by system in terms of the potential for:

   —Reduced costs
   —Increased revenue
   —Improved patient care

4. An assessment of the risks of automation including:

   —Development and operating costs
   —Vendor performance capability
   —Change to hospital policies and procedures

5. The ranking and prioritizing of identified projects of value
6. A determination of an affordable level of expenditure considering:

   —Available dollars
   —Contention for resources
   —Regulatory pressures
   —Value of the opportunities/benefits to be achieved

7. The screening of the commercial availability of system packages to meet the hospital's opportunities/benefits and to:

   —Develop requests for proposal as required
   --Firm up cost data
   —Integrate projects

8. Selection of the most appropriate approach based on the criteria of:

   —Early return on investment
   —Funding future development costs through the achievement of cost savings, as quickly as possible
   —Low risk

9. The definition of the techniques by which progress can be measured to help assure the achievement of

the benefits by:

—Setting measurement benchmarks for each approved project
—Establishing benefit realization plans
—Establishing a project reporting system

The result of strict adherence to the described approach should be the development of plans that would keep costs and benefits of new systems in parallel as much as possible, while additionally insuring early return on investment and the opportunity to modify the plan without losing the economies identified. It is this kind of sound business analysis that identifies the attendant risks while clearly measuring achievement that is going to make certain hospitals successful in their data processing programs.

A resource allocation and planning system for the development and operation of health care delivery systems

by BERNARD W. BISE

Peat, Marwick, Mitchell & Company
Washington, D.C.

INTRODUCTION

Hardly a week passes when one does not hear or read about the crisis in American health care. To the consumer and the health and medical care provider, there is a growing recognition of the inadequacies inherent in the distribution, delivery and financing of this Nation's health care. The majority of Americans today rely on the independent physician for medical care, consulting an array of specialists for specific problems and utilizing the hospital as the predominant facility in which to receive treatment. Medical care is purchased primarily on a fee-for-service basis when acquired by an individual.

Our present fragmented system—or better, non-system—of medical care has evolved such that only the acutely ill patient may enter into the medical care system and then only if he has the health insurance or personal finances to cover the cost of his care. Insurance notwithstanding, the person whose illness requires extended hospitalization and/or treatment by sophisticated procedures and technology will almost certainly experience long-term financial indebtedness, if not bankruptcy. Individuals who believe themselves ill and/or those who wish to remain healthy will find access to the system difficult if not impossible.

That a comprehensive health and medical care program for all Americans can be financed and delivered through existing mechanisms is not the issue. The addition of resources—manpower, facilities, and money—to the present medical care environment will not alleviate the crisis. The system must be reorganized and restructured to meet the various levels of health and medical care needs of the American public.

During the past few years and probably for the first time in modern history, there has been an attempt—by private industry, consumer groups, and the Federal government—to reverse the trend of increasingly costly conventional medical care. Generally, the innovative delivery systems emerging allow the health care planner to examine carefully the real and perceived needs of the health care consumer. These alternative systems apply resources at appropriate levels, so as to be fully responsive to needs of both the consumer and the provider.

Many of these new alternative health care delivery organizations have entered the planning stage of development and a few have become operational. Regardless of the differences in their organizational structure, health and medical care benefits program, and provider relationships, certain common characteristics among these organizations have emerged. The most prominent stems from the underlying concept that financing care and delivering care are interrelated activities which are best served when coordinated within the same organization. Health care planners are becoming involved in a new business in which questions extend beyond medical practice. The planning and effective management of health care delivery require the application of a broad range of business skills and tools—marketing, production, finance—to supplement the delivery of medical care. Recent experience has shown that, indeed, a failure to devote adequate attention to the financial implications of program decisions has resulted in either total financial failure or significant reduction in the operational scope of the delivery system.

As is true for most new business enterprises, the new delivery systems have experienced great uncertainty in attempting to make realistic operating projections. There is scant historical data on which to forecast either potential enrollment or operating cost parameters. The health care planner must be able to examine the financial implications of alternative decisions made during strategic planning. Faced with such uncertainties, PMM&Co. has developed a computerized analytical system—Resource Allocation and Planning System for the Development and Operation of Health Care Delivery Systems (RAAP)—which may assist the health care planner in projecting potential operating outcomes, resulting from management decisions on potential target beneficiary populations, health and medical care resource utilization, requirements, allocation, and cost. The final output of this system is a pro forma cash flow statement which estimates the future financial condition of the delivery system, resulting from the service demand, resource utilization, and allocation proposed by the health care planner.

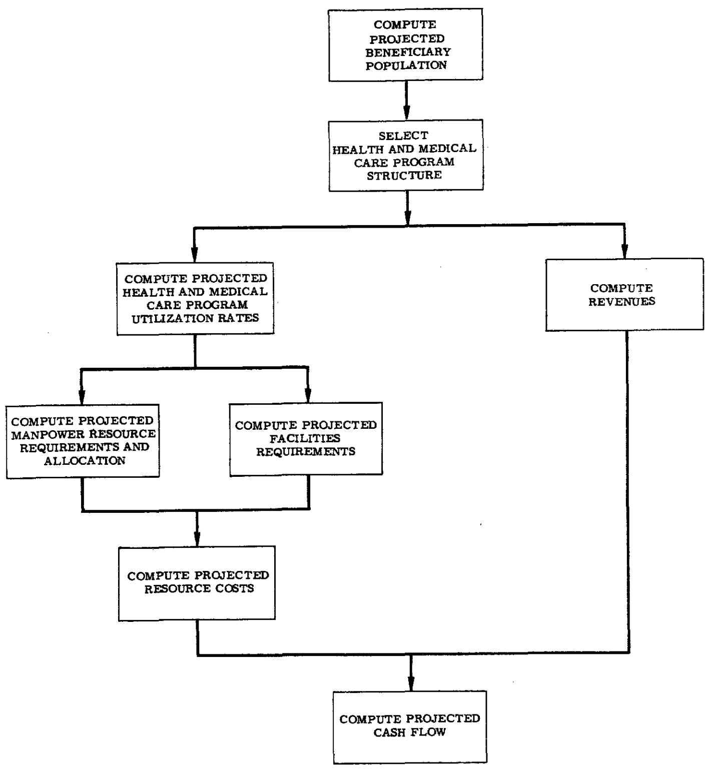

COMPUTE PROJECTED BENEFICIARY POPULATION
SELECT HEALTH AND MEDICAL CARE PROGRAM STRUCTURE
COMPUTE PROJECTED HEALTH AND MEDICAL CARE PROGRAM UTILIZATION RATES
COMPUTE REVENUES
COMPUTE PROJECTED MANPOWER RESOURCE REQUIREMENTS AND ALLOCATION
COMPUTE PROJECTED FACILITIES REQUIREMENTS
COMPUTE PROJECTED RESOURCE COSTS
COMPUTE PROJECTED CASH FLOW

Exhibit I—RAAP system framework

RAAP SYSTEM DESCRIPTION

The Resource Allocation and Planning System (RAAP) is a computer-assisted analytical system used by the health care planner to forecast or project operating results under a variety of operational assumptions and to reforecast as operating experience grows. Heretofore, developing projections of this kind was a long and painstaking effort; consequently, many organizations failed to examine all meaningful alternatives because of the tedious computations required.

The focal point of the RAAP system is a discreet timestep computer model. A "model" may generally be defined as the body of information gathered about a system for the purpose of studying the system. Since the purpose of the study is to determine the nature of the information gleaned, there is no unique system model.

The task of deriving a system model may be divided broadly into two subtasks: (a) establishing the model structure and (b) supplying the data. Establishing the structure determines the limits of the system and identifies the variables, attributes, and activities of the system. The data provide the attributes' values and defines the relationships involved in the activities.

PMM&Co. believes that both short- and long-range viability of a health care delivery system is dependent on management's comprehension of the relationship between utilization forecast and financial planning. The majority of financial and operational decision-making must be based on the delivery system utilization forecast. PMM&Co. further believes that resource requirements and the allocation of those resources to provide the health and medical care services and benefits is in large measure a dynamic process. The projection of maximum health and medical care program utilization and the application of resources necessary to meet that demand during the initial period of operations is likely to lead to significant negative cash flows which the delivery system might find difficult to overcome in subsequent years. We believe that resource requirements and allocation should be scheduled, based upon projected incremental monthly controlled utilization. This is not to say that there may not be a requirement for initial capital outlay for program start-up. However, dynamic resource utilization based upon incremental monthly projected resource utilization demands will aid in the establishment of a financially viable program during the initial phases of operation.

The RAAP system is structured around this concept. Exhibit I illustrates the logic flow of the model framework. Resource utilization and allocation, as well as operating expenses and revenue, are estimated based upon expected program utilization by the potential beneficiary population. To develop the pro forma cash flow projection the system requires quantitative data on the size of the target population and expected health and medical care program use-rates by the beneficiary population. The combination of population size and use rates determines probable resource requirements and allocation and the resultant cash flow.

One of the primary values of RAAP is its rapid examination of alternative strategies which need to be considered during the initial planning periods. Most factors used in the system can be readily modified to meet differing assumptions, and the impact of these assumptions can easily be computed. When the health care planner has developed a formalized operational and organizational structure, the RAAP system will project likely operating results against which the planner can measure the delivery system actual performance. The system is an effective management planning tool and also permits the delivery system to analyze systematically alternative decisions and evaluate operational experience.

The RAAP system operates in a "time-sharing" mode, a concept which allows one computer to process several user applications simultaneously, resulting in lower individual cost. The RAAP system has been designed such that it can be used directly from the health care planner's office merely by using a portable computer terminal and a common office telephone. The operating mode of RAAP is "conversational" in the sense that specific assumptions which the planner may wish to vary (i.e., salary rates, hospital per diem rates, program utilization rates, etc.) are entered each time the program is run permitting rapid comparison of alternative assumptions and corresponding financial and operational implications.

Some of the potential analyses that may be made by RAAP are:

- initial estimation of operating results;
- analysis of the impact of varying premium rates;

- analysis of the impact of varying beneficiary population size or market penetration;
- analysis of the impact of rising hospital costs or changing hospital utilization;
- analysis of the impact of alternative service delivery structure;
- analysis of the effect of including a dental care program; and,
- analysis of the impact on manpower requirements and costs of program structure.

One of the major values of the system is its capability to highlight the costs associated with gradual staff increases proportionate to service demand, as opposed to the hiring of a fixed "start-up" staff. When used in conjunction with other management information, the system can evaluate the impact of current trends and their effect on staffing and costs.

Effective use of the RAAP system requires the health care planner to develop a set of operating assumptions which are used as inputs to the system. The information used to develop these operating assumptions usually emanates from preliminary studies of the potential beneficiary population and available health and medical resources. In general terms, the types of information required to formulate system assumptions are:

- demography of potential beneficiary population;
- expected level and rate of program utilization;
- nature, scope and potential benefits package;
- expected health and medical resources use rates;
- expected levels of compensation to health and medical care providers; and,
- expected organizational and operational structure.

Based on the input assumptions outlined above, the RAAP system produces 15 output reports. The user has the option of choosing the number and order of printing of these reports in accordance with his particular needs. The reports are as follows:

- projected monthly market penetration;
- projected family mix of beneficiary population;
- projected health and medical care program utilization;
- projected space requirements;
- projected manpower requirement, allocation and cost by programs of care;
- projected management and administrative manpower requirements and cost;
- projected long-term debt service obligation; and
- projected pro forma cash flow statement.

Perhaps one of the more important benefits of the RAAP system is its requirement for an initial set of formal input information. The input information will require the planner to make certain decisions relating to the delivery system's organization and operation. Obviously, this is a most difficult task and requires much research, data collection, and analysis. The RAAP system provides a formal approach to such an effort. Having established the initial data bases of information, the RAAP system allows the planner to study the impact of alternative decisions in terms of resources and cost. The projected cash flow analysis alone is not a sufficient budgeting tool, but it does require the development of resources and cost data that are key elements and form a basis for a system that can be used to control program cost.

RAAP SYSTEM PLANNING APPLICATION

PMM&Co. realizes that the design and implementation of a comprehensive health care delivery system necessitates decision-making in many areas, such as organizational structure, legal and tax status, benefits package design, staffing, and marketing. Our experience suggests that unfortunately the financial implications of decisions made in these areas appear to have been ignored, unrecognized, or improperly emphasized, ultimately resulting in relatively substantial losses. Entrepreneurs have learned to prevent financial catastrophies by basing operational decisions on sound business judgments. Health planners now need to adapt and apply their concepts of prevention to the body fiscal (management) as well as the body physical (subscriber) for ensuring the solvency of the former permits continued service to the latter.

The RAAP system is applicable in the planning of, and projecting the likely operating results of, many types of health care delivery systems. Systems such as:

- ambulatory care centers;
- group practices;
- hospital ambulatory care departments;
- medical foundations; and,
- health maintenance organizations

to name a few, might use the RAAP system to assist the health care planner in the development of programmatically and financially sound delivery systems.

During the previous two years, Peat, Marwick, Mitchell & Co. (PMM&Co.) has been assisting HMOs in developing operationally sound and financially viable programs of health and medical care delivery. A major portion of our assistance has been projecting potential operating revenues and costs.

It is the Health Maintenance Organization (HMO) which appears to hold the greatest promise of providing the required elements of a new and responsive health care delivery system. The HMO concept envisions an organized and well-managed system providing a comprehensive package of health and medical care programs and services which shift from a disease/hospital-care orientation to a prevention and ambulatory care approach. This concept implies an obligation to render family-oriented health and medical care with an emphasis on continuity of care from preventive through rehabilitative services, on patient education and counseling, on ambulatory diagnosis and treatment, and on legitimate and expeditious referrals to appropriate specialists. The HMO's fundamental economic distinction is its method of payment—revenues based upon coverage of the voluntarily enrolled population served rather than reimbursement of fees to providers for services to consumers.

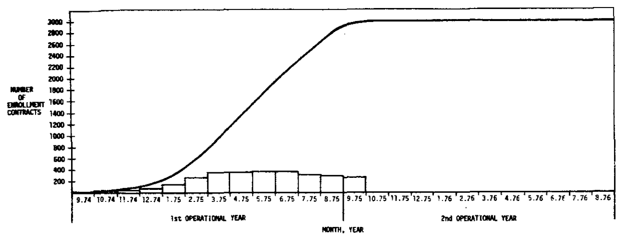


Exhibit II—Projected enrollment growth curve

The following pages present a sample application of the RAAP system to a hypothetical HMO. The sample application will illustrate the development of the required input data elements necessary to compute the likely resource requirement and cost, and the projected pro forma cash flow statement. For the purposes of this example the necessary research, data collection, and analysis of the target population has been completed. Adjusted to the findings and conclusions, benefits package and price structure have been developed. The HMO planner also has developed an alternative organizational and operational structure. The planner now wishes to assess the financial viability of his program.

Lacking any actual operational data for the proposed HMO, the planner must develop a set of operational assumptions which are used to project the likely HMO operating results. The assumptions pertain to:

- the HMO's enrollment level and growth rate;
- the HMO's resources utilization rate; and,
- the HMO's resources capacity and cost.

In each of these three areas the following assumptions were made.

Enrollment Level and Growth Rate

- Experience of newly-formed HMOs suggests that a penetration of 10 percent of the potential target beneficiary population during the first year is not unrealistic. Accordingly, the planner assumes a 10 percent penetration of the employees of the major employer in the identified target area. Accordingly, an enrollment population of *employed* family contracts, distributed as 20 percent single-member families, 20 percent two-member families, and 60 percent three-or-more member families with an average family size of 3.5 people/family are assumed.
- An operational start-up date of September 1974, with a two-year experimental operational period.
- An initial enrollment population of 10 percent of the expected total enrollment is projected with growth at a rate based on a cumulative normal distribution growth, which allows for: (a) a slow build-up phase, first three months; (b) an exponential growth phase, next seven months; and (c) a leveling-off phase, the last two months of the first year, with a steady state population after one year of operation. (Exhibit II)

Resource Utilization Rates

- Resource utilization (i.e., manpower, money, facilities and equipment) are directly determined by enrollment population on a *month-to-month basis, without anticipatory staffing.*
- An average ambulatory care use-rate of 8.7 ambulatory care visits per person per year (based on an average of Kaiser-Portland Health Plan and OEO data reported in *Medical Care,* 10:187-200, May-June 1972), with 4.2 visits/person/year for general health and medical care programs, 2.0 visits/person/year for mental health programs and 2.5 visits/person/year for dental health programs.
- A hospital use rate in patient day per 1000 enrollees based upon HMO's estimate of expected use, and at a cost represented by the average daily billed charges of the hospital.
- Prescription drugs provided at an average rate of four (4) prescriptions per member per year at an average cost to the plan of $3.00 per prescription.

Resource Capacity and Cost

- Professional manpower requirements for ambulatory and in-hospital care programs under a "team approach" will be met by one or more "physician teams": (a) composed of one full-time equivalent (FTE) physician and 1.5 FTE paraprofessionals (physician's assistant or nurse practitioner); (b) servicing 35 patients per day

```
CLIENT NAME?
ENROLLMENT TABLE NAME?TEST3

BENEFIT PACKAGE COMPONENTS
MEDICAL SUPPLEMENTAL-YES=1?1

DENTAL CARE-YES=1?0

AMBULATORY TEAMS-YES=1?1

PERCENT 1,2,3+ PERSON FAMILIES?20,20,60

3+ PERSON FAMILIES, AVG. NO./FAMILY=?3.5

CAPITATION RATE FOR 1,2,3+ PERSON FAMILIES?18.62,44.99,65.42

AVG. ANNUAL PATIENTS DAYS/1000?700

HOSPITAL PER DIEM?110.00

WHAT IS TARGET ENROLLMENT?7400

WHAT IS STARTING ENROLLMENT?740

WHAT IS PLANNED PERIOD STARTING MONTH AND YEAR?09,74

WHAT IS PLANNED PERIOD ENDING MONTH AND YEAR?08,75

REQUIRED REPORTS?1
?2
?3
?4
?5
?10
?11
?12
?13
?99
```

Exhibit III—Projected resource requirements and cost—10 percent market penetration

```
TARGET ENROLLMENT BY MONTH
 MO YR      TARGET   INCREMENT    1-PER.    2-PER.  3+PER.
10 74        851.       111.        170       170      510
11 74        962.       111.        192       192      577
12 74       1140.       178.        227       227      683
 1 75       1495.       355.        298       298      896
 2 75       2105.       611.        421       421     1263
 3 75       2938.       832.        587       587     1762
 4 75       3770.       832.        754       754     2262
 5 75       4603.       833.        920       920     2761
 6 75       5435.       833.       1087      1087     3261
 7 75       6201.       766.       1240      1240     3720
 8 75       6845.       644.       1369      1369     4107
 9 75       7400.       555.       1480      1480     4440


ENROLLMENT BY TYPE OF FAMILY
 MO YR ENROLLMENT  **************TYPE OF FAMILY**************     INCOME
       CONTRACTS   1-PERSON   2-PERSON  3-PERSON       TOTAL    DOLLARS
10 74     851.       170.       340.      1787.        2298.     44230.
11 74     962.       192.       385.      2020.        2597.     49999.
12 74    1140.       228.       456.      2393.        3077.     59230.
 1 75    1495.       299.       598.      3139.        4036.     77691.
 2 75    2105.       421.       842.      4421.        5684.    109421.
 3 75    2938.       588.      1175.      6169.        7932.    152689.
 4 75    3770.       754.      1508.      7918.       10180.    195958.
 5 75    4603.       921.      1841.      9666.       12428.    239226.
 6 75    5435.      1087.      2174.     11414.       14675.    282494.
 7 75    6201.      1240.      2480.     13023.       16743.    322301.
 8 75    6845.      1369.      2738.     14375.       18482.    355762.
 9 75    7400.      1480.      2960.     15540.       19980.    384608.
```

Exhibit IV—Project HMO subscriber enrollment

(8-hour session); (c) compensated on the basis of $45,000 per year per FTE physician and $18,000 per year per FTE paraprofessional, including all fringe benefits and all in-hospital care services; and (d) available to trust participants on a 16-hour per day basis (8:00 a.m. to 12:00 midnight, or 2 sessions per day).

- Professional manpower requirements for ambulatory and in-hospital consultation/referral care programs will be met by approximately 0.20 FTE physicians/1000 par-

```
PROJECTED UTILIZATION OF HEALTH & MEDICAL RESOURCES
 MO YR M # PERSONS   *********AMBULATORY CARE VISITS********* HOSPITAL VISITS
       D  ENROLLED     GENERAL     MENTAL     DENTAL       TOTAL   AV PT DAYS
10 74 M     2298.        804.       383.        0.         1187.        134.
      D                   40.        19.        0.           59.          4.
11 74 M     2597.        909.       433.        0.         1342.        152.
      D                   45.        22.        0.           67.          5.
12 74 M     3077.       1077.       513.        0.         1590.        179.
      D                   54.        26.        0.           79.          6.
 1 75 M     4036.       1413.       673.        0.         2085.        235.
      D                   71.        34.        0.          104.          8.
 2 75 M     5684.       1990.       947.        0.         2937.        332.
      D                   99.        47.        0.          147.         11.
 3 75 M     7932.       2776.      1322.        0.         4098.        463.
      D                  139.        66.        0.          205.         15.
 4 75 M    10180.       3563.      1697.        0.         5260.        594.
      D                  178.        85.        0.          263.         19.
 5 75 M    12428.       4350.      2071.        0.         6421.        725.
      D                  217.       104.        0.          321.         24.
 6 75 M    14675.       5136.      2446.        0.         7582.        856.
      D                  257.       122.        0.          379.         28.
 7 75 M    16743.       5860.      2791.        0.         8651.        977.
      D                  293.       140.        0.          433.         32.
 8 75 M    18482.       6469.      3080.        0.         9549.       1078.
      D                  323.       154.        0.          477.         35.
 9 75 M    19980.       6993.      3330.        0.        10323.       1166.
      D                  350.       167.        0.          516.         38.


 AMBULATORY CARE FACILITY SPACE REQUIREMENT
      1 SQ. FT./ENROLLEE=  19980.
     1.5 SQ. FT./ENROLLEE=  29970.


     PROJECTED MANPOWER REQUIREMENTS FOR AMBULATORY CARE

 MO YR AV VISITS  TEAMS  *******FULL TIME EQUIVALENTS*******   MANPOWER
        PER DAY  REQ'D  PHYSICIANS  NURSES PARAPROF    TOTAL   EST COST
10 74     59.4    1.7         1.7     2.3      2.5      6.5     11949.
11 74     67.1    1.9         1.9     2.6      2.9      7.4     13508.
12 74     79.5    2.3         2.3     3.1      3.4      8.7     16002.
 1 75    104.3    3.0         3.0     4.0      4.5     11.5     20989.
 2 75    146.8    4.2         4.2     5.7      6.3     16.2     29561.
 3 75    204.9    5.9         5.9     7.9      8.8     22.5     41251.
 4 75    263.0    7.5         7.5    10.1     11.3     28.9     52940.
 5 75    321.0    9.2         9.2    12.4     13.8     35.3     64629.
 6 75    379.1   10.8        10.8    14.6     16.2     41.7     76319.
 7 75    432.5   12.4        12.4    16.7     18.5     47.6     87073.
 8 75    477.4   13.6        13.6    18.4     20.5     52.5     96113.
 9 75    516.2   14.7        14.7    19.9     22.1     56.8    103906.
```

Exhibit V—Ambulatory health and medical care program utilization projections

```
          PROJECTED REFERRALS

MO YR # PERSONS   FTE - MD.    COST OF
        ENROLLED  REFERRALS   REFERRALS
10 74    2298.       0.5        1532.
11 74    2597.       0.5        1732.
12 74    3077.       0.6        2051.
 1 75    4036.       0.8        2691.
 2 75    5684.       1.1        3790.
 3 75    7932.       1.6        5288.
 4 75   10180.       2.0        6787.
 5 75   12428.       2.5        8285.
 6 75   14675.       2.9        9784.
 7 75   16743.       3.3       11162.
 8 75   18482.       3.7       12321.
 9 75   19980.       4.0       13320.


          PROJECTED HOSPITAL CARE

MO YR # PERSONS   HOSPITAL    HOSPITAL
        ENROLLED      DAYS       COSTS
10 74    2298.      134.0     14744.
11 74    2597.      151.5     16667.
12 74    3077.      179.5     19744.
 1 75    4036.      235.4     25897.
 2 75    5684.      331.6     36474.
 3 75    7932.      462.7     50897.
 4 75   10180.      593.8     65320.
 5 75   12428.      724.9     79744.
 6 75   14675.      856.1     94167.
 7 75   16743.      976.7    107436.
 8 75   18482.     1078.1    118590.
 9 75   19980.     1165.5    128205.


          PROJECTED MANAGEMENT & ADMINISTRATIVE EXPENSE

MO YR # PERSONS   MANPOWER    MGMT&ADMIN
        ENROLLED  REQ'D-FTE        COSTS
10 74    2298.       6.4        5750.
11 74    2597.       7.3        6500.
12 74    3077.       8.6        7700.
 1 75    4036.      11.3       10100.
 2 75    5684.      15.9       14225.
 3 75    7932.      22.2       19850.
 4 75   10180.      28.5       25474.
 5 75   12428.      34.8       31099.
 6 75   14675.      41.1       36724.
 7 75   16743.      46.9       41899.
 8 75   18482.      51.7       46249.
 9 75   19980.      55.9       49999.
```

Exhibit VI

ticipants, compensated on the basis of $50,000 per year per FTE physicians including all fringe benefits, roughly distributed as follows:

Otolaryngology	0.04 FTE
Dermatology	0.03 FTE
Ophthalmology	0.03 FTE
Radiology	0.03 FTE
Orthopaedics	0.02 FTE
Clinical Pathology	0.02 FTE
Neurology/Neurosurgery	0.01 FTE
Anaesthesiology	0.01 FTE
Special Surgery	0.01 FTE

- Professional Manpower Requirements for program management and administrative services will be met by supplementing 2.8 person/1000 private sector partici-

```
          PROJECTED REVENUES & EXPENSE


          PROJECTED REVENUES

SOURCE              QTR 12 74  QTR  3 75  QTR  6 75  QTR  9 75      TOTAL
SUBSCRIPTIONS         153458.    339801.    717678.   1062671.   2273608.
 CONTRIBUTIONS
 OTHER
TOTAL REVENUE         153458.    339801.    717678.   1062671.   2273608.

          PROJECTED EXPENSES

AMBULATORY CARE        41458.     91801.    193888.    287092.    614240.
DENTAL CARE                0.         0.         0.         0.         0.
HOSPITAL CARE          51154.    113269.    239231.    354230.    757884.
REFERRAL CARE           5315.     11768.     24855.     36803.     78741.
PHARMACY               12197.     27008.     57043.     84463.    180711.
MGMT.&ADMIN.           19950.     44174.     93298.    138147.    295569.
G & A COST              6643.     14710.     31069.     46004.     98426.
FACILITIES & MAINT.
DEBT SERVICE            4604.     10194.     21530.     31880.     68208.
EXCESS LIAB. INS.       9208.     20388.     43061.     63760.    136416.
TOTAL EXPENSES        150528.    333313.    703975.   1042380.   2230196.


          PROJECTED CASH FLOW

NET CASH FLOW           2930.      6488.     13703.     20290.     43412.
```

<u>STEADY STATE PHASE</u>

```
          PROJECTED REVENUES & EXPENSE


          PROJECTED REVENUES

SOURCE              QTR 12 75  QTR  3 76  QTR  6 76  QTR  9 76      TOTAL
SUBSCRIPTIONS        1153823.   1153823.   1153823.   1153823.   4615291.
 CONTRIBUTIONS
 OTHER
TOTAL REVENUE        1153823.   1153823.   1153823.   1153823.   4615291.

          PROJECTED EXPENSES

AMBULATORY CARE       311718.    311718.    311718.    311718.   1246871.
DENTAL CARE               0.         0.         0.         0.         0.
HOSPITAL CARE         384615.    384615.    384615.    384615.   1538460.
REFERRAL CARE          39960.     39960.     39960.     39960.    159840.
PHARMACY               91708.     91708.     91708.     91708.    366833.
MGMT.&ADMIN.          149997.    149997.    149997.    149997.    599988.
G & A COST             49950.     49950.     49950.     49950.    199800.
FACILITIES & MAINT.
DEBT SERVICE           34615.     34615.     34615.     34615.    138459.
EXCESS LIAB. INS.      69229.     69229.     69229.     69229.    276917.
TOTAL EXPENSES       1131792.   1131792.   1131792.   1131792.   4527168.


          PROJECTED CASH FLOW

NET CASH FLOW          22031.     22031.     22031.     22031.     88123.
```

Exhibit VII—Projected pro forma cash flow initial build-up year

pants, compensated on the basis of 13 percent of program revenues.

- An annual general and administrative (G&A) rate of approximately $10,000 per 100 participants per year to, cover such items as equipment rental, utilities, maintenance, insurance, supplies, travel, etc.
- A debt service rate of 3.0 percent of program costs to cover initial capital outlay costs.
- Purchase of excess liability insurance to absorb losses in excess of 125-150 percent of income at a cost of 4 percent of income.
- An actuarially determined monthly capitation structure of $18.62, $44.99 and $65.42 for one-person, two-person and three-plus person families, respectively.

The allocation of resources required to meet the anticipated health and medical care program utilization by the projected enrollment for the recommended delivery system are presented in Exhibits III through VII.

For the HMO, the major input assumptions are presented in Exhibit III. These assumptions include the family mix and size; capitation rate; average anticipated hospital use rate and charges; projected enrollment target; and the enrollment level to be achieved prior to the HMO going operational.

Exhibit IV illustrates monthly and cumulative enrollment targets based on the overall growth target pattern set by the user, and the number of individuals covered by the HMO based on the monthly enrollment targets and the percentages of two, three, or more person families. The report also shows monthly income based on the capitation rates provided by the user.

Exhibit V illustrates the projected ambulatory health and medical care program utilization by the enrolled population. The number of expected monthly and daily visits for general health and medical episodic care and mental health care are presented. The projected monthly hospital patient-days are also presented, together with the average daily census. The projected ambulatory care space requirements are also presented. The projected manpower requirements by category are presented with their cost.

Exhibit VI projects the number of full-time equivalent physicians required for specialist referrals and the corresponding projected costs; the estimated in-patient hospital days and the associated per diem rate costs; and the projected manpower requirements for management and administration of the HMO and their associated costs (as a percentage of revenues).

The results in Exhibits III through VI represent projected operating results for the initial build-up year. The resource requirements for each month of the steady, stable period would be the data presented in the twelfth month of the build-up year.

```
       PROJECTED REVENUES & EXPENSE


       PROJECTED REVENUES

SOURCE                  QTR 12 74  QTR  3 75  QTR  6 75  QTR  9 75        TOTAL
SUBSCRIPTIONS             153458.    339801.    717678.   1062671.     2273608.
 CONTRIBUTIONS
 OTHER
TOTAL REVENUE             153458.    339801.    717678.   1062671.     2273608.

       PROJECTED EXPENSES

AMBULATORY CARE            41458.     91801.    193888.    287092.      614240.
DENTAL CARE                    0.         0.         0.         0.           0.
HOSPITAL CARE              52981.    117314.    247774.    366882.      784951.
REFERRAL CARE               5315.     11768.     24855.     36803.       78741.
PHARMACY                   12197.     27008.     57043.     84463.      180711.
MGMT.&ADMIN.               19950.     44174.     93298.    138147.      295569.
G & A COST                  6643.     14710.     31069.     46004.       98426.
FACILITIES & MAINT.
DEBT SERVICE                4604.     10194.     21530.     31880.       68208.
EXCESS LIAB. INS.           9208.     20388.     43061.     63760.      136416.
TOTAL EXPENSES            152355.    337358.    712519.   1055031.     2257263.


       PROJECTED CASH FLOW

NET CASH FLOW               1103.      2443.      5159.      7639.       16345.
```

Exhibit VIII—Revised pro forma cash flow

Pro Forma Cash Flow Projections

Having developed needed resource requirements and their costs, there remains then, finally and most importantly, the question of assessing the financial viability of the HMO's operation as developed by the planner. The net results of the revenue and expense projections computed in the preceding pages are displayed in a single pro forma cash flow statement, Exhibit VII. The computations of cash flow (income less expenses) is a simple and straightforward process. Net cash flow is computed monthly over the planning period and is displayed quarterly for the planning period. This report shows projected subscription revenues and blank lines for other income. It also shows cost projections by program area—ambulatory care (team or nominal), dental care (optional to the user), referrals, hospital care and management and administration.

The net positive cash flow generated by the hypothetical HMO suggests that the alternative delivery structure proposed by the planner is a financially viable option. Because any forecast is subject to uncertainties, projections summarized for the HMO are not represented as specific results which will be obtained, but rather as operating results which can reasonably be expected under the assumed conditions.

The RAAP system as presented in this document cannot guarantee the long- or short-term success of an HMO. However, the system provides the HMO planner with a formal procedure to assist the planner in understanding the relationship that exists among the proposed program of health and medical care benefits, utilization patterns, resource requirement, and costs. In addition, when the planner has formally established the HMO's organizational and operational structure and entered operations, the RAAP system provides a baseline against which actual operational results can be measured.

RAAP SYSTEM PERFORMANCE APPLICATION

When the health care planner has formally established the delivery system's organizational and operational structure and management has commenced operations, the delivery system's actual performance must be continuously monitored. A delivery system's ultimate success or failure is determined by its actual performance. Therefore, it is essential that health care planners develop effective performance monitoring and evaluation criteria/mechanisms.

The RAAP system has direct application in health care delivery system performance evaluation. The operational projections produced by RAAP for the finalized delivery system structure represent an expected performance benchmark data base against which actual performance may be measured. Provided all the operational and utilization assumptions used by RAAP hold true, the delivery system's actual performance must meet or better those projected by RAAP if the operation is to remain financially viable. If actual performance falls below that projected, corrective measures must be instituted. During the replanning process, RAAP is used to assess the impact of the contemplated operational changes. The revised operational projections now serve as the new benchmark against which to measure actual performance. It should be emphasized that RAAP is not a management reporting system but works in conjunction with it.

As an example of the performance or management application of RAAP, let us return to the hypothetical HMO presented in the previous section. The HMO's monthly financial reports show hospital cost exceeding the projected expenses. The RAAP system projected hospital costs based upon a use rate of 700 patient days per 1000 enrollees at a fixed cost of $110.00 per day. The HMO's program utilization monitoring system reveals that actual hospital utilization is 750 patient days per 1000 enrollees. Hospital utilization review along with enrollment studies reveal that the increased hospital use rate is not excessive. HMO management must now determine if the HMO's fixed revenue from monthly subscriber payments will be sufficient to cover the cost of increased hospital use rates. Exhibit VIII presents a new pro forma cash flow based upon the revised hospital use rate. The revised cash flow suggests that even with a decrease in net cash flow of $27,000, HMO revenues are still sufficient to cover expenses.

Medical data processing in the United States

by MARION J. BALL*

Temple University Health Sciences Center
Philadelphia, Pennsylvania

In the past few years, there has been a sharp rise in the demand for health care services of all types by a larger and increasingly more well-informed public. At the same time, the cost of delivering high quality care is becoming formidable for hospital and patient alike. The introduction of the computer and information management techniques is considered by many informed health professionals and knowledgeable leaders in the medical computer field to be a solution to some of the major health management problems we are confronted with today.

Such systems should not be expected to magically solve all health care delivery problems; computerized systems can, however, alleviate much of the congestion in the medical communications network. It has been stated that the lack of an effective means of communication between health care professionals is one of the most serious drawbacks to improved patient care today.

Increased use of medical computerization upgrades this communication. Additionally, it assists the physician to carry out duties which he does not like involving documentation and clerical functions. It effects a reduction in personnel workload by taking over much of this paper work.

Computerization also reduces errors. It is imperative to good health care that the possibility of error be minimized. To ensure accuracy in the transmission and storage of data for subsequent use, a computerized medical information system can establish consistent standards and continuously monitor all transactions. All entries that require verification can be immediately printed at the entry terminal and verified by the user, thereby providing positive confirmation of the data. A printed record of the transaction is then available for future reference.

The computerized medical information system also promotes an important organization and accessability of valuable medical information. The physician can have direct access to all stored information through the use of remotely located terminals. A timely patient summary report provides accurate and convenient information supporting on-going medical care as well as enabling the implementation of a more exact system of capturing charges and giving provider credits. Of course, legibility is most certainly a positive benefit as well.

This type of system also improves physician communication with the community medical facility through incorporation of a so-called hospital information system. Through this system, an improvement of overall hospital organization and procedures is effected. Control is obtained by supplying the staff with concise information concerning events as they are occurring. The system can, therefore, serve as a tool in making day-to-day as well as long-range operating decisions. A major advantage of computerization is in the improvement effected by the systematic organization of medical functions and patient information in an accessable data bank. An information system establishes specialized files containing data on various functional activities and on patients. This data is distributed throughout the system for appropriate uses. The data base is constantly being updated during daily operation.

It is important to keep in mind that the principal advantages derived by physicians and patients from the installation of a computer system in the hospital occur after the physician has initially seen the patient. These include services subsequently executed after examination. Advantages affecting patient care include more rapid and accurate means of ordering tests and medication and of reporting, filing, and retrieving test results. While a computer system may not immediately reduce the time required to perform a specific test, the ordering and notification system will be speeded up.

There appears to be some confusion as to just what constitutes the medical information system. When used correctly, the term implies fully computerized management information as well as aspects of patient care. Although a comprehensive total system does not presently exist, the beginning of an ultimate system is being explored. In order to produce this type of system, an integrated flow of data processing will be necessary to cover:

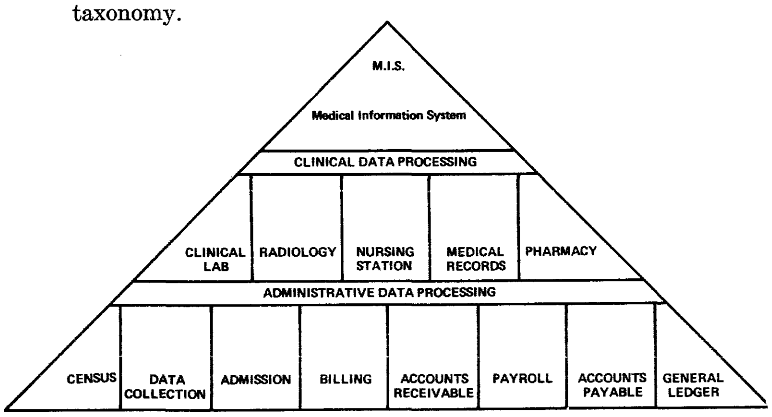
1. the collection of source data
2. the transmission of information via terminals and other communication links to a central computer
3. the establishment of a large, immediate-access data bank
4. the development of computerized management information to be used for decision making
5. the unification of the physician and his staff with an on-line, real-time computer complex

* This paper appeared in the journal "Hospital Financial Management," January 1974, Vol. 28, No. 1, pp. 10-30.

An all-inclusive system will facilitate communications between the activities of the various areas involved. To achieve acceptance of this system, the entire approach to our current methodology must get a face lifting. It is appropriate at this time to quote Machiavelli in reference to attempting this feat.

"It must be remembered that there is nothing more difficult to plan, more doubtful of success, nor more dangerous to manage, than the creation of a new system, for the initiator has the enmity of all who would profit by the preservation of the old institutions and merely lukewarm defenders in those who would gain by the new ones."

Before beginning to consider the field of Computer Medicine which involves the study of the various computer assisted medical applications making up the potential community medical information system, one needs a frame of reference within which to consider the current status of its development. The following diagram offers such a taxonomy.



Some of the applications mentioned in the diagram are discussed below.*

1. *Admissions and Bed Control:* Prepare current census report. Monthly periodic statistical reports. Bed status and accounting. Generation of lists of current patient locations and condition for use by physicians, clergy, telephone operators, etc. Message transmission to and from other pertinent areas (admissions, nursing, etc.). Assistance in patient admission interview. (For a small hospital—too costly.)
2. *Billing and Accounts Receivable:* More than 1,000 hospitals have one form or another of data processing. The biggest problem is capturing the charge information and entering the charge against the right patient. Most use batch processing. General ledger and budgeting usually are computerized after billing, accounts receivable, payroll/personnel, and accounts payable. These are most useful to management and planning.
3. *Payroll/Personnel:* This can be considered by the 100-300 bed hospital. These vary in sophistication, i.e., FICA and federal income tax deductions, to more complex problems with bonds, insurance savings, etc. The more sophisticated the more the cost; you get what you pay for.
4. *Accounts Payable:* This is a very early and justifiable computer application. Again, as in the foregoing areas, the repetitive nature of the task and the use of the same data to produce numerous reports makes this function a natural for the computer. As a result, it is usually automated early along with patient billing, accounts receivable, and payroll.
5. *Purchasing:* This is justified if combined effort is made with pharmacy, food service, maintenance, and central purchasing. If each is considered alone, it is not cost justified. Because of a lack of cooperation, coordination, and control in most institutions, this area is not one of the first to computerize.
6. *Inventory Control:* This can reduce the physical inventory if combined with a computerized purchasing system.
7. *Maintenance/Engineering Department:* Programs that can be considered in this area area: (1) preventive maintenance scheduling, (2) work order request control and costing, and (3) personnel scheduling. These programs can be undertaken on a variety of systems and have no bearing on the type of HIS or computer facility your hospital has.
8. *Laboratory Medicine:* All subfunctions relating to the discipline of Pathology including clinical pathology, gross and microscopic specimen analysis (also cytology), and forensic pathology. Not only are the commonly automated areas of hematology and serum chemistry included but also urinalysis, histopathology, and microbiology. It also encompasses quality control, trend analysis, laboratory instrument monitoring, and all aspects of specimen control including intradepartmental file management techniques when applicable. Laboratory is quite successful; has small dedicated facilities. No reduction in cost but smaller increase of cost as rate of growth increases. Charges generated and reported on lab system which helps the business office. A lab doing over 300,000 tests a year should consider a lab system.
9. *Scheduling:* All subfunctions involved with the heavily transaction-oriented aspects of patient care concerned with bringing together certain resources (men, money, material), various facilities (including admissions and discharge functions), and patients in a logical manner most appropriate to completing the diagnostic work-up. Scheduling of inpatient and outpatient appointments is a part of this function.
10. *Administrative Management:* Within the hospital, this includes all subfunctions involved with cost accounting, patient accounting, bed availability, bed usage, medical staffing, medical material, budgeting,

* National Center for Health Services Research and Development: *Comprehensive Hospital Computer Applications Program.* Vol. 1. *A Guide to Automation for Hospital Administration.* Rockville, Md., 1972.

programming (including preventive maintenance) and facility usage and planning. For the doctor's office, it also includes all management functions including the filling out of third party payment forms and patient billing. It also includes areas of health planning and the development of models.

11. *Pharmacy:* This must be considered in conjunction with other systems such as in-house accounting or shared hospital accounting for cost justification. The systems currently available do pricing, charging, inventory control, formulary listing, purchase order preparation, and other administrative type applications. All subfunctions relating to therapeutic agents and their usage. These include drug inventory control, prescription formulation, maintenance of a formulary, and appropriate aspects of clinical pharmacology such as usual dosage, orders checks and reminders (inpatient), contra-indications, and hypersensitivity reactions. In addition, appropriate aspects of clinical toxicology and acid-base (fluid) therapy are included as subfunctions.
12. *Radiologic Diagnosis and Therapy:* Work is being done in administrative applications, radiological information systems, and radiographic diagnostic systems. All subfunctions relating to Diagnostic Radiology including the conversion of requests into facility and technician schedules, patient preparation notices and reminders systems for reporting on individual roentgenograms, and maintenance of a film file locator. In addition, all subfunctions relating to therapeutic radiology are included such as the calculation of isodose curves, patient treatment schedules, and tabulation and maintenance of the results of treatment (Tumor Registries).
13. *Multitesting and Health Screening:* This is controversial and dependent on the decision of the professional staff. All subfunctions relating to history taking and physical examination separate from and included within areas making up multiphasic health screening functions. It also includes all outpatient methods of acquiring the basic data base on the patient alone or in conjunction with a physician's assistant.
14. *Nursing Services:* All subfunctions related to the scheduling and prompting of direct nursing care functions including shift-change summary reports on the patient's course and condition. In addition, a summary of all doctors' orders on demand and any additional nursing administrative functions not covered specifically in other areas is included. Computerization of the nurse's reporting (notes) system itself is also included. This is useful for visiting nurses as well as those based on the doctor's office or hospital.
15. *Patient Support:* All subfunctions involved with patient comfort including adjuncts to nursing care. It includes request processing, delivery of service, and inventory control in usually hospital based areas such as inhalation therapy, occupational therapy, and physical therapy as well as the central supply. It also includes the facility Food Service incorporating some or all of menu planning, nutritional accounting, and control of all foodstuffs through procurement, inventory, stockroom issue, food production, and daily cost accounting.
16. *Patient Monitoring:* All subfunctions involved with the integration and presentation of large amounts of physiometric data by sophisticated instrumentation frequently interfaced directly with the computer. Such efforts include those usually found in intensive care units (general, cardiac, burn), surgical recovery rooms, delivery rooms, and emergency rooms.
17. *Medical Records:* This is costly and difficult—not really a consideration for a 100-300 bed hospital. If 20 or more records per day are retrieved, strong consideration should be given to computerizing diagnostic and operative indices. This function refers primarily to advanced record management functions including prospective and retrospective statistical analysis. It is based on the patient record file system which is an intrinsic development of all the various functional areas themselves. It includes medical file folder inventory control procedures as well as individual reporting (summary) procedures. It also includes coding functions often associated with such documents, statistical compiler development, and other systems for on-going correlation of useful medical information based on such parameters as diagnoses, therapy, and tracking the clinical progress of disease (i.e., Epidemiology). Doctors' consultative reports (including summaries, operative reports, and correspondence) are also included under this category.
18. *Electrocardiography (EKG):* Here there is a possible cost reduction which the hospital administrator can investigate. When the computer is used correctly, the cardiologist can increase his work capacity at times tenfold by the use of computers. When one of the 27 EKG regional facilities is used, the average cost is three to four dollars per EKG, which includes the EKG equipment, terminals, communications, and computer time.
19. *Electroencephalography (EEG):* No general acceptance. It is hoped that when a new theory of statistical probability is acceptable, better than 75 to 80 percent accuracy will be available using computer interpretation, on abnormal versus normal readings leaving only 20 to 25 percent to be interpreted by the physician.
20. *Respiratory Therapy:* In most cases, part of an overall HIS and not a dedicated therapy system. Programs deal with the administrative function, physiological monitoring, and multiphasic screening, i.e., collecting and evaluating spirometry data, blood gas analyses, and some diagnostic programs. These programs are found in both time-shared systems and dedicated minicomputers, however, not in small community hospitals.
21. *Physiological Monitoring:* These systems are comprised

of on-line instruments and patient monitoring equipment with warning signals should the conditions exceed pre-defined set bounds. Careful record of the patient's progress is kept. Monitoring is seen in intensive care units, coronary care units, operating rooms, delivery rooms, surgical recovery rooms, nursery areas, pulmonary care units, and catheterization laboratories.

22. *Food Service:* Food service personnel and menu planning computerization. Dietician and cooking staff is largely manual, although new equipment has made automation more efficient and cut costs. Computer Assisted Menu Planning (CAMP) is a package to plan a series of varied menus with predetermined nutritional requirements at minimal costs. This then leads to the planning for purchase of raw food, and control over inventory. Disadvantage is that initiating CAMP is very expensive and can offset the benefits gained. In most installations, one year of the dietician's time was required to set up the system.
23. *Diagnostic Support:* All subfunctions in direct support of the physician's diagnostic work-up. This includes request processing, delivery of service, recording results (sometimes automatically), allocation of resources for such elements as electrocardiography, pulmonary functions, and nuclear medicine, and computer-assisted diagnostic consultation.
24. *Medical Library:* All subfunctions involved with bibliographic referencing, the maintenance and updating of computer displays on important parameters, and other appropriate medical library functions.

Hospital communication systems have been devised to tie many of the above functions together. A brief description of each of the most popular approaches is discussed below:

1. The *Nursing Station Approach,* one of the most widely represented views, maintains that activity at the Nursing Station is most directly related to patient care.
2. The *Medical Record Approach* can serve as a total, uniform, fiscal, historical, and medical data base. A standardized system of recordkeeping must be instituted in order to computerize effectively.
3. The *Fiscal Approach* to computerized Hospital Information Systems is to establish a data base at the admitting office.
4. The *Multiphasic Screening Approach* is based on the theory that taking a patient through an extensive series of pertinent examinations before he is admitted to the hospital makes it possible to establish a data base for subsequent use.
5. The *Research-Oriented Approach* is the least represented, and hopefully, in the future, the medical industry will devote additional efforts in this area. Here the emphasis is on Epidemiological information rather than on current hospitalization data.
6. The *Modular Approach* is becoming more and more popular due to the emphasis on dedicated turn-key systems such as in the laboratory, pharmacy, business office, etc. The concept here is to establish a well-planned linkage to a central depository out of which the communication system evolves and grows.
7. *Distributive Systems Approach* centers around the development of a network of integrated stand-alone systems which interact.

Several computer companies have entered into the field of computer based health care systems; some of the major ones are discussed below and on the following pages.

BURROUGHS/MEDI-DATA, INC.

Functions: Clinical Laboratory, Patient Support, Diagnostic Support, Medical Records, Scheduling, Pharmacy, Administration and Management.

Burroughs Corporation offers Medi-Data Hospital Information System in support of a stand-alone system (complete) for a single hospital or a group as a service from the Charlotte Data Center, Charlotte, North Carolina.

Medi-Data, Inc., has developed a hospital information system in which the nursing station is the nucleus of activity. The system provides information processing and message-switching capabilities in the following three areas: (1) patient care, (2) administrative accounting, and (3) research and statistics. On-line data terminals, consisting of a CRT unit, keyboard, and card reader, can be connected to the central computer by means of leased telephone lines or hard wired as appropriate, i.e., can be hospital's own system or remote. Duke University, North Carolina, has just installed this system.

The Burroughs/Medi-Data system now automates many hospital routines including: (1) complete census, pre-admissions, admissions; (2) transmitting doctors' requests and communications to the appropriate ancillary departments; (3) nurses' clerical-service functions; (4) hourly departmental scheduling, e.g., operating room, X-ray, hospital maintenance and housekeeping, etc.; (5) transmitting test results and/or diagnoses to the proper location; (6) recording and updating data on the patient's medical record; (7) floor stock resupply; and (8) all fiscally-oriented procedures.

Action is initiated via the physician's hand-written orders. The ward secretary, located at the terminal, then enters through the keyboard all patient-related information, This data is displayed on the CRT screen, and a hard-copy is also produced for verification. Upon its confirmation, the data is sent to the central processing unit for permanent storage. The central processing unit generates all care and medication schedules for each nursing station, pharmacy, laboratory, department, etc.

Test results from ancillary departments are entered directly into the computer, and they are automatically printed out at corresponding nursing stations to be included in the patient's chart. All entries are validated by the computer to confirm the contents by patient number and by doctor's identification number. Laboratory results are summarized daily (on new activity) and printed in descending date order by patient by procedure for the chart.

Outpatients can be entered into the system through an outpatient registration system similar to that of inpatients. Medical orders can be entered in the same manner as inpatients' data. Requisitions are then generated to applicable departments, and charges are posted. Results may be sent to the requesting clinic or department. Outpatients may stay on the system for a predetermined period of time or as scheduled for repeat visits.

Burroughs hardware and support has facilitated Medi-Data (non profit) Hospital Information System. At present, Burroughs themselves are entering the hospital information systems field on their own corporate level with different terminals (plasma terminals) and a somewhat different approach than taken by Medi-Data. Such a system can be seen at Wesley Hospital in Wichita, Kansas.

CONTROL DATA CORPORATION

Functions: Clinical Laboratory, Patient Support, Diagnostic Support, Scheduling, Medical Records, Patient Monitoring, Pharmacy, Administrative Management.

Control Data Corporation's concept of an integrated medical information system is to coordinate all health related activities—patient care, administration, education, and research—via the MEDICOM System. MEDICOM, now in the design state, is a communication and patient-file management system which will form the basis for the integrated system. St. Louis University Hospital in Missouri is involved in this project.

In a typical system, it is planned that the computer will process selected patient data which are entered on-line from remote terminals. The central processing unit will store the data on active mass storage devices while the patient is in the hospital. Patient files will be organized into active and passive files. The active files will hold all the information available on a patient when he is receiving health services. When the patient is released from the hospital, the active file will be summarized, added to the passive file, and released to an inactive storage portion of the system.

Reports and requests to the computer center may be entered by conversion to card input, page readers, or directly through on-line entry/display terminals. CDC has developed and utilized the MEDISCOPE system to handle all hospital communications.

MEDISCOPE is a data acquisition and information retrieval system that is based on the CDC 1700 computer with optional links to the CDC 3000 and/or 6000/7000 computers. The specially designed CRT terminal is called the "Digiscribe" and includes 20 transport strips on the entry/display screen. By touching one of the 20 strips, the user can manipulate pre-programmed subject matter and/or enter unprogrammed material via the attached keyboard.

To assure patient privacy and security, the mode of identification for entry into the system is a combination code consisting of a number, color, and animal (e.g., 6, green, fox) to be keyed in at the terminal.

Currently, the dedicated systems pertaining to medical computer applications that are offered by CDC include: (1) MEDLAB, an intensive-care system; (2) CARDIOTEST, and EKG system; and (3) CLINLAB, a clinical laboratory management and automation system. Several of these applications can be seen at Latter Day Saints Hospital in Salt Lake City, Utah. The MEDISHARP business system includes the SHARP (Shared Hospital Administrative Report Processing) system.

Although all of the above systems have the capability of being stand-alone systems, when they are combined with MEDICOM, they will form Control Data's concept of an integrated medical information system. However, no such complete system is in operation at this time.

DIVERSIFIED NUMERIC APPLICATIONS

Planned Functions: Clinical Laboratory, Patient Support, Medical Records, Scheduling, Pharmacy, Radiologic Diagnosis and Therapy, Administrative Management.

Diversified Numeric Applications is developing HOSPITROL, a computer controlled data handling and communications system which will consist of centrally located computers (MED/16s). HOSPITROL is being designed specifically to utilize multiple computers for effectiveness and reliability; the computer was designed to be used in multiple computer configurations through the use of partitioned, shared core memory.

For example, in a typical 400 bed hospital, DNA's HOSPITROL system will employ three MED/16 processors in a minimum configuration (software is designed to obtain maximum utilization from this redundancy), four processors if total clinical laboratory automation (UNI-LAB) is included, or possibly five processors if complete accounting capability is desired. Six types of communication consoles will be combined in various ways to provide the appropriate terminal(s) at every ancillary department. Each of the units is described below: (1) Plasma/600 is a CRT unit for nursing station communications. The unit includes an alphanumeric keyboard for names and variable data, an "action" keyboard for initiation of various functions, and selection buttons for medical order information. The console also contains a badge reader for positive identification of the user. Plasma/610 and Plasma/620, modified versions of the Plasma/600, can be used in the departments of pharmacy and radiology, respectively. (2) Plasma/500 is a keyboard/display console with less display area than the Plasma/600. This console is designed for use in those ancillary areas where high-speed display and keyboard entry are essential, e.g., surgery, the delivery room, laboratories, etc. (3) Video/400 is a large CRT display console developed for use in admissions, the business office, medical records, information desk, and administrative departments. The unit contains a simplified keyboard for functions similar to that of a typewriter and also includes administrative function control keys. (4) Format/300 is a hard-copy printer that contains a specially formated keyboard for the individualized input requirements of each department it serves. The printer provides a continuous audit trail of all transactions, prints reports, etc. The Format/300 can be used in dietary, central supply, radiology, etc. (5) Labeler/200 is a printer designed specifically for printing labels which can be used in pharmacy, laboratory, etc. (6) Reporter/100, a printer designed for location in the emergency room, business office, nursing station, etc., will provide all hard-copy printout.

In addition to ordering, storage, routing, and checking data, HOSPITROL is being designed to provide patient treatment schedules; patient/departmental supplies ordering; patient charge handling; pre-admission and admission functions; access to medical records, billing data, and statistical data; etc. The prototype is being installed at North Memorial Hospital, Minneapolis.

EXECUTIVE DATA SYSTEMS, INC.

Functions: Clinical Laboratory, Scheduling, Nursing Services, Pharmacy, Administrative Management.

Hospital computer systems offered include both on-line and off-line types and cover the hospital areas of financial/administrative management systems, clinical systems, and systems' coordination. Among the system concepts followed are modularity and flexibility. Equipment used at the hospital varies depending upon the systems employed and the size of facilities. Some EDS hospital clients have no in-house computer equipment, while others have one or more pieces of computing equipment such as keyboards and keyboard-printers, card readers, magnetic tape units, line printers, cathode ray tubes, minicomputers, and even large-scale computers.

Of the operational systems, accounts payable, payroll, personnel, and physical plant are available without the hospital needing any in-house equipment. Hospitals choose from a range of in-house equipment including nursing station and departmental terminals when utilizing the other operational systems.

Basically, each hospital designates the applications which it wishes to use and how it wants to use them. Master files and specific system-design requirements are then defined, and the applications made operational. Typical processing units with related peripherals include: Burroughs 3500, BMR 6135, IBM 370/145, and IBM 1800. Auxiliary units include Burroughs N and Mohawk 2501. Terminals include IBM 1050, IBM 1092, Burroughs N, and minicomputers with CRTs (Datapoint 2200s). This hardware is used for a shared fiscal as well as a H.I.S. at St. Barnabus Hospital in Livingston, New Jersey.

System options which a hospital can follow are to choose to computerize payroll with no in-house equipment performing financial/administrative functions through the use of a shared computer network with in-hospital terminals or mini-computers or to include clinical subsystems with one or more in-hospital computers and multiple terminals located throughout the hospital.

The medical message-switching occurs through data entries at the terminals located at various areas in the hospital. Security codes allow for appropriate data entries and information retrievals. After editing, the data are processed according to their types and needs with message forwarding, file updating, and system interfacing occurring as appropriate. Currently, the departments operationally affected include admissions, business office, pharmacy, nursing, and laboratory.

HONEYWELL, INC.

Functions: Clinical Laboratory, Scheduling, Patient Monitoring, Pharmacy, Radiologic Diagnosis and Therapy, Administrative Management.

A typical intra-hospital message-switching system can consist of a Honeywell H-1015 with 131K of core memory, three disks, two tapes, a printer, a 35 G.E. Terminent, 300 R.O. terminals, and five Bunker Ramo CRTs with keyboards. It runs in 65K foreground of an operation system. This is implemented at Parkview Hospital in Ft. Wayne, Indiana.

The receive-only printers are located at the nursing stations, ancillary departments, admissions, P.B.X., kitchen, central supply, and pharmacy. One CRT is located in admissions, and the other four are situated in the data center. Patients are admitted on-line, and the ward, P.B.X., and other appropriate departments receive messages giving the patient's room, bed number, and physician's name. Admissions can query the system at any time for available bed census.

Nurses, doctors, emergency room, operating room, and other departments access the system by dialing "3" on a telephone. A CRT operator in the data center answers and is given the patient's name. The name is keyed into the CRT which displays the patient's name, account number, ward, room and bed number, and physician's name. This information is verbally verified with the caller. Services are ordered verbally, and CRT operators key in service codes. The service codes will bring up the service description so that the operator can verify the service ordered. When all orders are completed for each patient, the messages are released to the departments, and an audit message is printed at the nursing station for inclusion in the patient's chart. Where an order would involve several departments, for example, a G.I. series, all departments (i.e., X-ray, kitchen, pharmacy) would receive orders pertaining to it from the single transaction code for a G.I. series.

Upon a patient's discharge, finance, P.B.X., and housekeeping are notified from the single discharge code. Housekeeping dials "3" when bed and room are ready for occupancy. This notifies admissions of bed and room availability.

Presently, all transactions are spooled from a disk each night to the patient account system. It is planned to do patient accounting on-line. The system can produce a census on demand.

INTERNATIONAL BUSINESS MACHINES CORP.

Functions: Clinical Laboratory, Patient Support, Diagnostic Support, Medical Records, Scheduling, Patient Monitoring, Pharmacy, Radiologic Diagnosis and Therapy, Administrative Management.

IBM Corporation's concept of an integrated medical information system utilizes the System/360 or System/370 processor(s) to form the foundation for: (1) a communication system that centrally controls the flow of source data to ancillary locations throughout the hospital; (2) a central information system that electronically receives, transmits, and stores data for immediate access; and (3) a real-time system that processes and provides data in a desirable format.

IBM's medical systems are designed to assist many different service areas in the hospital. These include: nursing stations, admissions, pharmacy, clinical laboratories, X-ray, dietary, electrodiagnostics, operating room, central supply, medical records, business office, etc.

Doctors' requests for patient treatment, laboratory analysis, laboratory results, and other information are entered at terminals located at nursing stations and ancillary departments or by calling a central terminal pool. This information is stored in the central files of the computer, which uses random access disk storage for rapid reporting and large capacity. The data obtained from the central processing unit provides the basis for the preparation and printout of schedules, patient data, messages, and reports. This system can be seen at Layola and Little Company of Mary Hospitals in Chicago and at Monmouth Hospital in New Jersey.

Basic terminal configuration includes combinations of 2260 CRTs and 1050/1092s. The 1092 terminal is composed of an entry keyboard and printer connected to the central processing unit via communication links. It uses plastic overlays placed over the 16 columns of keys to identify the data being entered.

Each patient can be identified by the depression of a single key in the first four columns. This key is assigned to the patient upon his admission, and the patient's name is written on the overlay by the nurse. This patient overlay remains on the keyboard for all entries. The remaining 12 columns are used for entering specific orders for requests and are identified by a larger overlay. The terminals automatically sense the specific overlay being used, thus identifying the type of entry into the system. Groups of these 12 column overlays are kept at each terminal. In addition, the CRT terminal device, the IBM 3270, may be used in combination with the above-mentioned terminals. It includes a light-pen and badge reader.

McDONNELL DOUGLAS AUTOMATION COMPANY

Functions: Clinical Laboratory, Diagnostic Support, Scheduling, Pharmacy, Radiologic Diagnosis and Therapy, Administrative Management.

The McDonnell Douglas Hospital Patient Care (HPC) System is an on-line, real-time system designed to service the ordering, results reporting, and basic scheduling needs of the hospital. Originally installed in October, 1969, the HPC System currently services three hospitals located remotely to the data center, utilizing the shared computer concept. Original application areas installed were admissions, clinical laboratory, radiology, and nursing (ordering and patient information retrieval). Since the original installations, pharmacy and laboratory instrumentation modules have been installed and are currently operational.

The system is designed for ordering to be done from the nursing station with the clerical people on the station having the primary responsibility for order entry. Should the day arrive when physician order entry is required, the network, as currently installed, will require only a change in the input device to accomplish this. Utilizing cathode ray tubes as the ancillary entry devices, results reporting from laboratory and radiology are operational. Laboratory tests being done on SMA 12/60 and 6/60 are automatically retrieved through an on-line System 7 in one of the hospitals.

Currently, all ancillary departments and admitting use CRTs for entry devices with a push button keyboard at each nursing station. Development is presently underway to incorporate advanced terminals at the nursing stations. This system can be seen at St. Francis Hospital in Peoria, Illinois.

MEDELCO

Functions: Scheduling, Clinical Laboratory, Patient Support, Pharmacy, Radiologic Diagnosis and Therapy, Administrative Management.

The Medelco Total Hospital Information System (T*H*I*S) handles patient, room, and bed information on a real-time basis in addition to message-switching and storing financial transactions; it transmits orders and requests to and from nursing stations, automatically updates changes, updates inventory records by department, prints medication requests at the pharmacy terminal in label form, and handles all radiology, special diets, housekeeping, laboratory, and outpatient requests. Terminals can also be used as time clocks to collect payroll information. Terminals, consisting of a file of edge-punched cards; an optical card reader for input of data into the system; and a teleprinter for hard-copy output are located at each nursing station and every ancillary department.

Pre-punched cards for each order, service, or product available in the hospital are in the card files as are cards for patients. Since the central processing unit is a hard-wired machine with only a small amount of computing capability, these cards form the software. Desired cards are selected from the file to initiate a request. These cards are dropped through the card reader, and the combined data from the two cards is stored in the central processor and then transmitted to the appropriate department. The data is also printed out at the originating nursing station providing an accuracy check and a printed entry for the patient's chart. Other "stand-alone" computers such as on-line laboratory equipment, etc., can interface into the system.

The patient card is typed and punched when the patient is admitted into the hospital. This card goes with the patient to the nursing station and replaces the embossed identification plate. The order information cards are prepared by the manufacturer when the system is installed. They are kept at the nursing station in numbered pages in a visible file in open-book form. Pages are color-coded by department and alphabetized within the department. Each card is also numbered to correspond with its slot on the "page." This system can be seen at Deaconess Hospital in St. Louis, Missouri.

MEDAC*—Medical Electronic Data Acquisition and Control System designed and marketed by Metric Systems Corporation.

Functions: Patient Registration, Physician and Religion Lists, Bed Control, Housekeeping List, Census, Requisition Handling, Pricing, Timekeeping, Payment on Accounts, Current Charges, Inventory Control Information, Department Activity.

MEDAC allows for automatic input to the computer. This system is built according to the following distinct processing activities: (1) servicing of remote on-line terminals sending requisitions/messages to the system in a real-time mode; (2) communication with the central work station in a real-time mode and processing of basic system functions such as new admissions, bed status, and current charges; (3) execution of batch-mode programs such as census, daily charge activities, and department activity lists.

The system is designed in a manner that will allow expansion of the system functions without making change necessary to the basic functions. The following systems are independent subsystems, therefore, any combination of the functions may be included in the desired configuration: (1) bed control; (2) admitting and census; (3) printed requisitions delivered electronically; (4) automatic, uniform pricing; (5) computer input; (6) inventory control information; (7) timekeeping by work center; and (8) pharmacy package. This is the foundation of a modular approach.

Other features of the system are: automatic security check on all entries; instant validation of all entries; automatic message routing of all "Need-to-Know" locations; catalogues and prints lab requisitions by unit and type test; automatic generation of secondary messages such as "Hold Tray"; procedure and service summaries available by department; all transactions recorded on magnetic tape; allows plastic card or keyboard input at terminals; and all messages reflect date, time, and employee identification. This system is currently operational at Central Kansas Medical Center in Great Bend, Kansas.

* Purchased by B-D Spear on November 1, 1973.

REACH** (VITAL)

Functions: Blood Bank, Central Supply, Clinical Laboratory, Dietary, EKG, EEG, Scheduling, Medical Records, On-line Admissions, Pharmacy, Radiology, General Business Office Functions, and Overall On-Line Data Collection, Storage and Retrieval.

The REACH System (Real Time Electronic Access Communications for Hospitals) is a clinically-oriented medical information system. The CRT display terminals and associated printers are located at the nursing stations, in ancillary departments, and in the business areas. This affords hospital personnel the capability of acquiring both fiscal and medical information on a patient.

At the same time that the doctor or nurse enters a service request for laboratory work, X-ray, etc., a charge is automatically made on the fiscal record of the patient's bill and is recorded on the patient's medical record. In addition, as a by-product of ordering through the system, inventories are updated, and volume and statistical reports are automatically maintained for each department. Laboratory tests can be both ordered and scheduled from the nursing stations in order to make optimum use of the staff and facilities. Upon discharge of the patient, the medical record is stored on magnetic tape or hard-copy, and a complete bill is created for the patient.

The CRT display terminal includes 20 select buttons adjacent to the screen to indicate selection of data and an attached keyboard. These three sections—select buttons, CRT, and keyboard—comprise the input unit. Output is created by these same three sections plus a hard-copy printer.

The system identifies the user by a machine-readable badge, the size of a credit card, which is inserted into a slot on the upper right-hand side of the terminal. Upon insertion of the coded badge, the program will make available only the information which that particular badge entitles its holder to see.

The REACH System utilizes two NDC (National Data Communications), 1695 central processing units located in the hospital. The system offers the hospital service and back-up by using one processor on-line and one off-line in a batch processing mode.

National Data Communications, Inc., offers an off-line system called Fiscal Management Information System (FMIS) at a low cost in addition to the above complete system.

There are two real-time, front-end systems. The first is the Source Data Communications & Retrieval System (SDC&R) which can be implemented by application. An example is ADT (Admissions, Discharge, and Transfer).

** On October 1, 1973 Honeywell took over REACH, such that NDC's software will be licensed to the purchasing hospital.

This can front-end any present off-line the Hospital may have because all data is brought off on tape in ASCII.

At such time as the hospital wishes, it can add the Patient Medical Information System (PMIS). The REACH System can be seen at Deaconess Hospital in Evansville, Indiana.

SANDERS ASSOCIATES, INC.

Functions: Scheduling, Clinical Laboratory, Patient Support, Pharmacy, Radiologic Diagnosis and Therapy, Administrative Management.

Sanders Associates has developed the CLINI-CALL System, a computer-controlled, high-speed data system. It can extend to any area of the hospital-admissions, wards, surgery, pharmacy, dietary, laboratories, accounting, admissions, etc. In addition to storing, retrieving, sorting, and checking data, the system is designed to provide patient medical histories, current medical records, statistical summaries, and legal records. It will schedule medical tests, maintain inventories of beds, and maintain the current status of meal orders, accounting/billing records, etc.

All data entries and requests are made from data terminals located at strategic points throughout the hospital. The data terminal used consists of five modular units: (1) display unit, (2) data printer, (3) Photopen sensor, (4) keyboard, and (5) card reader.

The CRT unit includes an electronic Photopen device which is the size of a fountain pen. It is used as a pointer to select data and/or "action words" from the display screen. The operator at the terminal touches the indexed items he wishes to see on the base of the display screen with the Photopen unit. When the action word "execute" is activated by the pen, the index is immediately replaced with newly selected information. This information can then be studied, updated, modified, or erased by using the pen or a touch command at the terminal keyboard. The keyboard may also be used to enter new patient data and/or instructions in the central processing unit, call up selected stored records, and request special communications with other terminals in the system. The hard-copy printer produces paper records and labels on command from the central processing unit.

Personnel wishing to access the system are identified by inserting a small identification card in the card reader at the terminal. The reader determines the level of a user's authority and unlocks the system for his use. The system is also programmed to withhold specific data items from individuals who do not have specific authorization on their identification cards.

The central processor is the focal point of the CLINI-CALL System through which all data inputs, requests, and communications from the terminals must pass. System operating software is FOPS (File Oriented Programming System); it is designed by Sanders and allows for the writing of application programs in logical English Statements. This system can be seen at Kaiser San Francisco Hospital in San Francisco, California.

SEARLE MEDIDATA, INC.

Functions: Clinical Laboratory, Diagnostic Support, Scheduling, Patient Support, Medical Records, Multitesting and Health Screening, Pharmacy, Administrative Management.

The Searle Medidata Network 320 Hospital Information System is a modularly-expandable communications and applications computer system. It is designed to route orders, collect charges and other financial information, and return results directly to a nursing station. It is expandable through additional modules to meet specific applications: Lab 320 in the clinical laboratory, Pharmacy 320 in the pharmacy, and Financial 320 for financial data processing.

The heart of the system is a unique "Touch-Terminal." The terminal presents 320 patient order choices at one time to the operator on a single overlay sheet. A few such overlay sheets can hold all the orders for a particular station. An order is entered by touching the phrase (or phrases) describing the order on the "Touch-Terminal." The computer prints the order immediately at every "need-to-know" terminal including a confirming printout at the originating station. The printer operates at 30 characters per second.

A completely backed-up computer configuration is used with two identical computer systems installed at each site. All data are retained on two separate disks and are further logged on a magnetic tape for a third level of data security.

In addition to the "Touch-Terminal," other terminals may be attached to the system. Cathode ray tubes are used to input textual and extensive numeric data in the admitting office and the financial areas. A receive-only version of the "Touch-Terminal" is available for installation in areas where it is not necessary to enter orders; maintenance, housekeeping and dietary are typical sites. A low cost "Check-In" terminal is available for locations requiring only the capability for an employee to check-in or out for time-keeping purposes.

The availability of a back-up computer makes it feasible to implement some systems without adding an additional computer. For example, the back-up computer can run Multitest 320 (Automated Multiphasic Health Testing) during the day, and a complete package of financial programs (Financial 320) can be run at night.

The Network 320 system can also be shared among several hospitals. In order to add a hospital to an existing group, terminals and a controller are added to the hospital, and a set of look-up tables are prepared to meet the hospital's specific requirements. This system is being installed at Baptist Memorial Hospital in Jacksonville, Florida.

SPECTRA MEDICAL SYSTEMS

Functions: Scheduling, Clinical Laboratory, Patient Support, Diagnostic Support, Medical Records, Pharmacy, Administrative Management.

The Spectra-2000 System is an on-line, real-time data system which utilizes a dedicated computer and a large hospital patient data base. The facility is centrally located in the hospital and can be simultaneously accessed from any of the remote terminal stations in key areas of the hospital. Examples of these key areas include patient care areas, pharmacy, various laboratories, and administrative locations. Terminals located at remote stations include a multicolor video display for data entry and display, a light-pen and keyboard for data entry, and a printer for hard-copy output.

The system processes the following types of data: (1) admission information, (2) medical orders, (3) test results and narrative reports, (4) nurse-generated orders and comments, and (5) captured charges for services performed. Information is processed for patients from the time of pre-admission or admission through two days after discharge.

The information processing impacts patient care and hospital operations in the following ways: (1) medical orders are incorporated into reports on patient status showing active orders, test results, medications given, and results/reports outstanding; (2) requisitions are transmitted to the laboratory, pharmacy, diet kitchen, and other ancillary services; (3) scheduling capability is provided for medication administration and performance of laboratory tests; (4) various reports are produced to facilitate hospital operations, e.g., laboratory work sheets, specimen pick-up lists, specimen and medication labels, scheduled admissions list, beds available list, census and activity reports, etc.; and (5) control is provided for pharmacy inventory, automatic refill of medication orders, time-out of non-renewed medical orders, and clinical lab patient-specimen test result integrity.

The system may be accessed through entry of an appropriate identification code. The code is a unique six-character password, and only a valid password gains access to the system.

Entry of the physician's password causes the system to display his list of patients. The physician selects the desired patient by pointing the light-pen to the appropriate name and depressing the button on the shaft of the light-pen. Using the same technique, the physician then may review active orders or enter new medical orders for that patient by simply pointing the light-pen at key phrases and data that are assembled to create the orders. Only in exceptional situations will the physician use the keyboard. However, other users, such as the medical secretary, will use the keyboard primarily as the input device. At present, this system is being installed at Mary's Help Hospital, Daly City, California.

TECHNICON MEDICAL INFORMATION SYSTEMS CORP.

Functions: Scheduling, Clinical Laboratory, Patient Support, Medical Records, Pharmacy, Administrative Management.

Technicon (formerly Lockheed Information Systems) has developed MIS-1, a regional medical information system in which the nursing station is the center of activity. Using special I/O terminals at each nursing station, physicians and nurses initiate the data base.

Admission data are entered to MIS-1 through the Video Matrix Terminal (VMT), thus initiating the collection and processing of data. Once a patient's file is created, the data entered are available for automatic compiling and rapid retrieval by the physicians, nursing staff, and other authorized personnel.

Physicians enter orders and review the patient's chart directly through the VMT at the nursing station or by telephone to the nursing station. Orders are transmitted automatically to the appropriate departments creating a printout of requisitions and instructions in all ancillary areas. Orders are also easily recalled through the VMT for review by nurses and the physicians.

The VMT display terminal consists of three basic parts: the display screen, a light-sensing pen, and a keyboard. The user can enter, extract, or manipulate information primarily by using the light-pen and secondarily by using the keyboard.

While Technicon's business office system can function independently of MIS-1, it is designed to be an integral part of the total system. Upon entry of patient information, a computer record file is established for the purpose of accumulating billable transactions. This data is obtained directly from the original record of information by a hospital staff member.

Users can gain access to MIS-1 by entering a valid identification number through the keyboard. MIS-1 allows access to stored information only in accordance with the predetermined rules applying to each type of user.

Technicon recommends the use of a regional data processing center where several hospitals can utilize the IBM 370/145 central processing unit. The Technicon central processor stores all patient data, processes inputs, directs all VMT terminal and printer operators, and prepares reports for the various hospitals using the MIS-1 system. El Camino Hospital in Mt. View, California, has installed this system.

OTHER SYSTEMS AVAILABLE

There are many other companies either entering into or already committed to marketing hospital or medical information systems. Several companies have been marketing dedicated systems in various areas of the hospital, e.g., laboratory admitting, radiology, EKG analysis, pharmacy, etc.; and, having been successful in these specific areas, some companies are planning to integrate dedicated systems through a message-switching system, to creating a unified, centralized hospital information system. Companies falling into this category include Digital Equipment Corporation, Medical Data Systems Corporation, General Electric's Medinet Systems, and Biometric Computer Service, Inc. (BCSI).

Companies involved in various phases of hospital information systems not discussed above are ITT, Medical Scientific International, Univac, Medical Information Technology

(MEDITECH), Shared Medical Systems, Medlab, Inc., Automated Systems Corp., Datacore, Inc., Medilogic Corp., National Cash Register and Standard Register. Many consulting organizations are entering into this field and will be making additional contributions in the years to come.

SUMMARY

In order to employ advanced communication's concepts in the various areas discussed in this presentation, the scope and nature of each task must be carefully defined. Choice and implementation of a computer system must be preceded by a thorough analysis of the present manual operation. The administrator and his colleagues must familiarize themselves with basic data processing principles. Without sufficient knowledge of how a computer system works, they are poorly equipped to compare available alternatives.

Other important considerations include realistic cost assessments and the quality of the hospital's in-house educational program. The dedication of the medical staff and the cooperation of the administration are essential.

ACKNOWLEDGMENTS

I would like to thank Dr. Michael Jenkin, Mr. James Petters, Mr. Gary Hammon, Dr. Stanley Jacobs, and the various vendors discussed for their help in preparing this paper; I would also like to thank Susan Bertolette for her editorial assistance. Further material of interest can be found by reading *How to Select A Computerized Hospital Information System* by Marion J. Ball (S. Karger, Basel, Switzerland, 1973).

Why industry won't hire your graduates

by GARY B. SHELLY

Anaheim, California

Vocational or career oriented education as used in this paper is defined as that type of education, at less than a baccalaurete degree, that is aimed at preparing an individual for employment in industry in a specific occupational area. The remarks included in this paper are directed at instructors, administrators, and other educators concerned with this area of data processing education, whether it is in the high school, the vocational school (public or private), the community college, and, in some instances, the universities. During the past year, while serving as an educational consultant in computer education, I have had the opportunity to contact over 400 schools throughout the United States and Canada with career oriented programs in data processing and frequently the comment was heard "What can be done to improve our program to make our students employable?" "Why won't industry hire our graduates?"

The basic answer to this question is that students in many data processing educational programs are not trained well enough in data processing and computer programming skills to enable them to be hired in industry at the conclusion of their schooling. It is perhaps helpful to explore the history of this phase of education in order to illustrate some trends of thinking and practice which have developed in data processing career education.

Vocational or career oriented data processing education had its earliest beginnings in approximately 1958 when community colleges began offering two year programs in data processing with emphasis upon the operation and control panel wiring of punched card or unit record equipment including the key punch, sorter, accounting machine, collator, reproducing punch, interpreter, and calculator.

In the early 1960's, as industry changed from punched card unit record data processing, to card oriented computer systems, schools attempted to implement vocational training in computer programming by utilizing the IBM 1620 computer system in their instructional programs. Although there were some practical reasons for utilizing the 1620 computer, such as the educational contribution which was made available by IBM and the fact that a FORTRAN compiler was available so that the computer could be used for mathematical programming, the training in computer programming on the 1620 computer did little to meet the needs of industry because the primary business machine used in this period of time was the IBM 1401 and its related family of computers. The training in computer programming in 1620 machine language certainly failed to stimulate industry's interest in public education and contributed little in meeting industry's needs for 1401 programmers with a knowledge of SPS, autocoder, tape processing, or IOCS. Thus, education failed to accept or even recognize the need and responsibility to provide the type of programmer personnel needed by business in this critical age of expansion within our industry. Industry could not turn to public education at that time to truly meet its need for trained personnel.

Later, in 1964, with the announcement of the third generation of computer equipment, there was an almost frantic search in industry for programmers knowledgeable in Assembler Language, in COBOL, in RPG, in something called job control and operating systems, and in multiprogramming, yet we find many schools as late as 1969 and 1970, and perhaps even now, teaching control panel wiring, 1620 computer programming, or perhaps programming using the IBM 1130 computer.

It should be quite evident from these trends in data processing education that proper up-to-date training has not been available from the beginning. If teachers and schools, in 1964, had the insight, knowledge and ability to react to the immediate needs of industry, and were turning out knowledgeable assembler language programmers with an adequate understanding of job control, or knowledgeable COBOL programmers, I feel confident that the service to the industry would have been invaluable and, in fact, saved industry thousands of dollars in training costs. Educational institutions must react rapidly in order to provide a viable means of career education in data processing.

Educators cite many problems—budget restrictions preventing installation of modern computer systems, no one is available to teach such subjects, etc. I can only reply that if education wants to serve industry, and if industry is going to hire graduates from these programs, then education must be able to meet the personnel needs of industry by teaching "state of the art" material. Anything less is doing a disservice to the taxpayers who are supporting an obsolete program and perhaps even more important, is wasting a tremendous amount of human resources by doing an injustice to the students who receive training in these obsolete areas. Today, career or vocational programs should be teaching the techniques and methods of programming in a virtual storage environment, programming for data communications applications, teaching data base concepts and methods of structured programming for business applications. How many schools are teaching these subjects now? The answer is very few, and if this does not change, then career educational programs are going to continue to fail industry by not providing the types of personnel which are required to be immediately productive on the job without additional industry training.

Certainly one of the arguments which education will use to justify the fact that their curriculum is not up-to-date is that the personnel are not available to teach the required subjects. Although I know of no research documenting the background of teachers in vocational data processing, my own acquaintance in the field of education tends to indicate that most teachers teaching data processing are former business education teachers who often have had a few college courses in data processing or computer programming. Most of these teachers have had little or no industrial background or experience either as computer operators, programmers, or systems analysts. A recent study by a major publishing company indicated in high schools the majority of teachers teaching data processing had been teaching one to five years. The next highest percentage had been teaching less than one year. The point that teachers are not available to teach required subjects seems to be a realistic appraisal of data processing teaching profession. The only response that I can give to this is that if our educational institutions will not recognize the need for constant upgrading of the faculty in this rapidly changing era of technology, and will not provide time for the instructors to upgrade themselves and to upgrade the curriculum by developing new courses, then there is very little chance that education will be able to train students for the needs of industry. Those in education must inquire as to what can be done in terms of curriculum development and teaching training and must provide the solution.

According to a National Science Foundation report, approximately 1700 institutions of higher education are now spending 500 million dollars annually for computer facilities and their operation. As a member of industry it is difficult for me to understand why a significant portion of these funds cannot be directed to the upgrading of teachers and curriculum, and even implementing curriculum when required, to meet the needs of the business data processing industry. Unfortunately, there appears to be a critical lack of leadership in career oriented data processing education on a national level. In attending computer conferences and reading data processing periodicals, I am constantly amazed at the millions of dollars of private and federal money which is being spent on computer-assisted instruction, terminals so that students can learn BASIC to solve problems in their business classes, and funding from the National Science Foundation to study the utilization of the computer in a variety of disciplines. The following are merely a sample of some of the fundings in computer activities which have been granted from the U.S. Office of Education.*

Development and Evaluation of Computer Assisted Instruction For Instrumental Music	$48,460.00
Development of a Computer Based Laboratory Program For Library Science Students	$104,480.00
Teaching Mathematics Through the Use of a Time Shared Computer	$185,421.00

Obviously, teachers in mathematics and other disciplines have expressed their need for funding to further their computer-oriented programs; however in the area of career education in business data processing, I have yet to see any national leadership emerge which has actively pursued, over a period of time, funding to develop curriculums, train teachers, communicate with industry, and perform other functions which are absolutely critical if career oriented data processing education is to fulfill its purpose. If vocational data processing educators are not willing to put forth the effort to attract the attention of those organizations which can be of assistance in these necessary areas, then I see little hope of vocational data processing ever reaching the point where it will fulfill the needs of industry.

This leads, then, to what really is required by industry. Although I cannot purport to be a spokesman for all of the business data processing industry, my experience as a business applications and systems programmer, a systems analyst, and a consultant for a software firm has given me some insights into the requisites of a student who will be successful as an employee in the data processing department of a company as an operator, a programmer, or systems analyst. Perhaps, the most obvious shortcoming of students coming from a career or vocationally oriented program is their lack of any in-depth knowledge of data processing, particularly as related to computer programming.

In speaking with many instructors and in reviewing the catalogs of schools which claim to train programmers for programming positions, the most common curriculum contains some type of introductory course with perhaps FORTRAN programming, a course in RPG, a course in COBOL, a course in Assembler Language, perhaps something in PL/1, finished off with a course in systems analysis and design where the major emphasis in on interviewing techniques, feasibility studies, and the management aspects of the systems study. In fact, one of the earlier studies relative to career education for third generation computers recommended that both COBOL and PL/1 be taught in a single semester course.

Unfortunately, then what industry too often finds when interviewing a student as a prospective employee is that he knows some FORTRAN, some COBOL, some Assembler Language and some PL/1 but does not have enough knowl-

* U. S. Office of Education Support of Computer Activities, January 1969 U. S. Department of Health, Education, and Welfare, Office of Education.

edge of any programming language to perform the duties required of a programmer trainee, such as maintaining programs, or writing relatively simple file processing programs. He cannot write a typical sequential file update program or create and process a realistic direct-access or indexed sequential file as utilized in industry. He does not have the ability to debug a COBOL or Assembler Language program with any kind of expertise by effectively reading the computer listing and related core dump. He has not been trained in the use of utility programs or manufacturer supplied Sort/Merge programs. Thus, when asked to prepare a job stream for a simple sort, the student is unable to do so simply because he has not been schooled in these basic fundamentals required of a productive programmer.

In discussing this problem with instructors, various explanations are heard for this lack of in-depth study. For example, one instructor remarked to me that "We don't teach any programming involving magnetic tape processing because we don't have the equipment . . . but we talk about it a lot." Another comment received was "Oh, if you can program for card input you can learn to program for tape and disk applications on the job." This attitude is one of the reasons education has failed—our educational institutions specializing in data processing should provide the student with in-depth training in a computer programming language which is widely used in industry within the area of the school, if the school is to meet the requirements of industry.

Other instructors have commented "We don't have time for in-depth training." Associated with this problem is the fact that students are often taught a great deal of subject matter that is not useful to them in attaining employment in the data processing profession. The prime example of this is the teaching of FORTRAN programming for vocational or career oriented students. Apparently, FORTRAN was taught in the early 1960's because it was the only high level compiler available on the widely used IBM 1260 computer and because it was one of the well-known languages. It is ridiculous, however, for career oriented business data processing educators to continue to teach FORTRAN because FORTRAN is not commonly used as a business applications programming language. The attempted justifications of various institutions such as it is an easy language to learn, or as indicated in one survey, all of the other schools are teaching it so we should also, is totally without foundation. FORTRAN should not be taught in a limited program which is designed for career or vocationally oriented students.

It is also apparent from examining curriculums and speaking with instructors that they are not aware or are not responsive to the critical needs of industry. Somehow the term computer programmer has become the magic word in career oriented data processing programs in both private and public institutions. There has been little attention given to training in areas where there is a critical need within industry—in training for computer operations, documentation, tape librarians, control clerks and key punch operators. According to a recent AFIPS survey, there are approximately 210,000 individuals employed as programmers, 200,000 persons employed as computer operators and 440,000 individuals employed as keypunch operators—these people all require training!

Many of the data processing instructors with whom I have spoken have acknowledged that only a small percentage of students entering career oriented data processing programs have the aptitude, intelligence, drive or motivation to succeed in industry as professional programmers. Many will agree, however, that most students could successfully operate a computer or become valuable employees in some capacity within the data processing department. Thus, it seems ironic that schools in their desire to serve industry have failed to realistically appraise the requirements of industry as related to the types of students which are entering their programs and the employment opportunities that are available to them. Certainly there is nothing disgraceful about a school training a computer operator who will earn a salary of $800.00 to $1,000.00 per month or a keypunch operator who will earn $500.00 to $600.00 per month. Yet there are few schools which are offering any training in computer operations and very few good, comprehensive courses in the area of keypunch training. These people are needed in industry. Why aren't they being trained in career oriented data processing schools!

Today, in industry, hundreds of thousands of dollars are being spent on in-house training. In many installations, this amounts to buying video-tape courses or P.I. courses from one of the many vendors who have come into existence within the past few years. It seems a disgrace to an industry which is striving to be professional that it must turn to video-tape courses costing $2,000.00 or more as a primary source of education. Yet, industry has no alternative. In addition to the "standard" courses in data processing available in video-tape form, one leading vendor in this area has teaching material and courses in Virtual Storage (VS) Concepts, VS Facilities, VS Job Control, VS Modular Programming, VS Utility Programs, VSAM, VSI Debugging, VS2 Debugging, Designing a Teleprocessing System, and other up-to-date subjects. There are few, if any, schools on any level that have similar courses available *now* to meet the immediate needs of industry.

The data processing industry has turned to in-house training, P.I. Courses and video-tape courses as a method of education because the schools which claim to teach data processing for vocationally oriented programs have not come close to supplying industry with quality students skilled in the knowledge and use of programming languages and data processing techniques who can become contributing employees when hired. Until this takes place, I am afraid that the only answer which industry can give to the question of "Why won't industry hire our graduates?" is "Because through career and vocationally oriented programs in our schools students have not been adequately trained to meet the needs of industry."

How the data processing industry has failed education

by THOMAS J. CASHMAN

Long Beach City College
Long Beach, California

Many attribute the greatness of the United States as an industrial nation to the system of private and public education which has evolved in our country. Those of us in education like to point with pride to the fact that our system of education is designed to provide each individual in our society with the opportunity to make maximum use of his abilities and to actively pursue the career of his choice whether on the semi-skilled, technical, or professional level.

It is indeed unfortunate that the business data processing industry has historically failed to recognize that formalized education can contribute to the success of their own operations within the data processing department, and, because of the importance of the data processing function within most organizations, the success of the company itself. When I speak of formalized education, I am speaking, not of manufacturer's training courses, P.I. courses, video tape courses or other types of multi-media courses offered as a part of in-house training, but formal educational programs of instruction in the form of well defined curriculums taught at our colleges and universities throughout the nation.

Almost each day one can read of the criticisms of the data processing departments of typical companies such as "Computers unproductive 48 percent of the time" or "Management becomes disenchanted with the computer," etc. Studies then point out that the failure of the computer installation is that top management did not get involved and similar excuses and explanations, but one seldom reads that the failure of computer installations is directly attributable to the lack of education at the programmer, systems analyst and data processing management level. When is the business data processing industry going to recognize that education, or the lack of education is severely handicapping the industry from progressing to the status which it deserves within the structure of the business organization?

Prior to 1971 a great deal of education within the computer industry took place at manufacturer training schools where an employee of a typical company could take a three to five day course in programming in a given language in which the instruction set was explained, and would then return to his company to program and perhaps even design a system! In recent years, in spite of the increased complexity of computer systems there has been an even greater deterioration of this vital function of education. In my own experience I have been acquainted with individuals who have completed a P.I. course in a programming language and have then been turned "loose" with no other supervision to reprogram existing applications. I have known other entire installations that have received their programming education via a video tape course and then set about to program their company's applications with their newly discovered skill in a new programming language. Certainly, if we can learn to program by means of a P.I. course, or by means of a video tape course of a few hours in length, and can indeed do an effective job, then programmers must be vastly overpaid!

At the other extreme we have the companies that insist that individuals entering the data processing profession, particularly as programmers, have a bachelor's degree. It makes no difference what area, but the person must have a bachelor's degree. I am sure that many of you are acquainted with the data processing manager that openly boasts of the fact that the newly hired programmer with the bachelor's degree in music is one of the best programmers on the staff —that is, after that individual had completed the company's six month training program and spent another year on the job. I am constantly amazed that the business data processing industry does not apparently recognize the tremendous direct cost associated with in-house training, and does not recognize that the business data processing industry should be able to recruit individuals trained in data processing at our colleges and universities rather than seeking individuals with majors in music or math but who have an aptitude for computer programming.

When discussing the extensive in-house training in data processing with training directors of several large corporations and those in management positions in data processing common replies have included, "but data processing is different than other areas—the field moves too rapidly", or "people must go through our training program to learn our system", etc.. When an inquiry is made as to whether they give P.I. courses or video tape courses in basic areas to their newly hired accountants, or electronic technicians, or their engineers, or chemists, the answer is almost universally "No, we hire these people from our colleges."

As one analyzes the fact that public education trains personnel successfully in many areas and disciplines why has education not been successful in meeting the needs of the business data processing industry as evidenced by the extensive in-house training in all phases of data processing? Although industry may chide those in education for not offering the courses that are needed, an equal responsibility must rest with industry for their lack of interest in formal education offered through our colleges. In the past fifteen years there has been virtually no leadership from the data processing industry through its related professional associations relative to curriculum development in career education in such areas as business applications programming, data processing management, etc.

In the meantime our colleges and universities continue to train computer scientists who can write compilers, but who are not making a substantial contribution to the day-to-day needs of the vast business data processing community that so desperately needs the programmer that can write an efficient COBOL program that will assist in getting the payroll out on time.

Certainly, the efforts of A.C.M. in the development and recommendation of a curriculum in information analysis and systems design is to be commended; however, "The program is intended for the education of individuals who will develop complex informations systems for organizations."* What about the training for business application programmers? According to a recent AFIPS report there were some 210,000 programmers employed in 1970 with a projected average need of 23,000 additional programmers per year through 1980. Are these individuals to be trained via P. I. courses?

The business data processing community has failed "education" in several areas: (1) Industry has failed to define on a national level what it wants in trained personnel. What are the requirements for a programmer? Industry blindly says "Programmer wanted, degree in any area required," others say "Programmer wanted, some college desirable—must score high on aptitude test". Industry says "coders are out", yet current research tends to indicate that programming teams with a chief programmer, a diagnostic specialist and a coder utilizing structured programming techniques is extremely effective. Certainly if education is to train for industry there must be some guidelines set as to the requirements for the positions for which the training is to occur. (2) The field of education as related to business data processing has on a career education level, had little support from our professional associations relative to curriculum development. If industry wants business application programmers to have a baccalaureate degree why have not our major professional associations actively sought to develop, recommend and implement four year degree programs in business application programming. If a four year degree is not required why have our professional associations not stated this fact. If the educational requirements of the profession are not known why have not our professsional associations undertaken research to determine the amount of education required to be successful within the industry. Historically, professional associations composed of a membership with a common bond of specialized knowledge have always been concerned about the education of its membership. Yet, little has been done within the business data processing industry to define the body of knowledge required to be successful within the profession.

Other professions such as law and medicine exercise great influence and control over the subjects taught in private schools and colleges related to their profession; however, business data processing seems uninterested or perhaps unwilling to let or even encourage our colleges and universities to enter this area of instruction. In an informal conversation with the director of education of one of the large professional associations related to data processing an inquiry was made as to why there was not a greater interest in developing and recommending a program of training for its members and taking further steps to encourage colleges to implement the program of studies within their curriculum. The reply was "This is a controversial area." Certainly, if industry cannot define the educational requirements for its members or cannot define what individuals entering the profession must know then it is extremely difficult, if not impossible, for educational institutions to offer an effective program of instruction.

Many working within the data processing profession including programmers, systems analysts, and data processing managers like to consider themselves "professionals." One of the commonly applied standards to determine if a field is a "profession" is that a "profession requires a high degree of academic training". How does this definition apply to the current status of training within the industry? At a recent speech by the president of one of our leading professional associations the point was made that an effort was being conducted to assist the membership in reaching top corporate management. The effort of the association to assist its membership to reach the lofty levels of corporate management was to supply a video tape course on management to be shown at local meetings! It is almost unbelievable that this is the solution espoused by a leading professional association to prepare those needing education in management for top management positions. Is this association unaware or unable to recognize that top management positions in most corporations require a high degree of academic preparation, normally as evidenced by a baccalaureate degree or higher? Has it not occurred to this association that perhaps a recommendation of a college level curriculum in data processing management might not be a more realistic approach to reaching the levels to which it aspires for its membership? Again, I mention this instance only to re-emphasize the fact that the business data processing industry seems unaware of the function that public education through our colleges can perform in preparing individuals for employment within their profession.

As previously pointed out other professions exercise great control and direction over instructional programs at all levels related to their profession. Recently, the health/medical technologies field has undergone dramatic changes

* Computing Newsletter, April 1972, Center for Cybernetics Systems, Colorado Springs, Colorado.

under the direction and guidance of related professional associations. There are now available in some states one year programs for licensed vocational nurses (LVN), two year programs for registered nurses (RN) and four year programs for registered nurses for those desiging to move into supervisory positions. Certainly these developments did not come about at a single school or by a single individual attempting to define the needs of the medical/health field but were brought about by the recommendations of professional medical societies. Why won't the professional data processing associations support education by defining a curriculum for colleges at several levels and exercising influence and pressure on our schools to provide the type of education needed so that the industry can elevate itself to a truly professional level?

It is interesting to review the development of commonly recognized professions and relate these developments to business data processing. Some of the steps leading to professional status have been defined as follows:

1. There is a body of specialized knowledge which becomes apparent and leads to the formation of a "professional" society or association bringing together those with common interests.
2. There is a sharing of knowledge of the members of the association. At this stage learning frequently consists of sharing of experiences, problems, and techniques. Apprenticeship type training and on-the-job training is provided for new entrants into the field.
3. The profession begins to recognize the need for more formalized training or education and seeks to define and recommend a program of training for those desiring to enter the profession.
4. As the body of knowledge related to the profession becomes more complex and theoretical, the responsibility for education is placed in the hands of the universities where the information can be widely disseminated and research relative to the field undertaken.
5. Licensure and certification of members is likely to occur especially where public interest or public service is involved.
6. The final step in the evolution of professions commonly consists of an interest by the profession in ethical problems of society and social responsibility as related to the profession.

As one reviews the current status of those in business data processing it becomes apparent that the industry is still in the "apprenticeship" stage in the evolution of the profession with a sharing of knowledge and in-house and on-the-job type training common in most installations. Isn't it about time that the business data processing industry, through our professional associations, began that important step toward professionalization by defining a body of knowledge that should be taught at all levels of education so that formal education can contribute to the success and growth of this dynamic profession.

A springboard for data processing education in Oklahoma

by DENISE A. PIERCE

State Department of Vocational and Technical Education
Oklahoma

Certainly the problems expressed in the preceding presentations have not been foreign to Oklahoma. Oklahoma has realized the vast need for a marriage between education and industry, particularly in the area of data processing where no traditional curriculum patterns exist and where there are no established career paths for data processing instructors to follow. Oklahoma had an additional need, however, the cooperation between the various State agencies in using a "system approach" to improve education, and more specifically data processing education. The existing bureaucratic machinery allowed a certain degree of amalgamation, but the gap had not been totally closed. This paper will seek to outline the framework of the cooperation between the State agencies, to discuss the state-wide teleprocessing system, to comment on the more usual types of involvements in data processing education, and close by discussing what Oklahoma is using as a springboard for accomplishing projects and meeting expressed needs.

As to the existing operating framework, one could view this as a triangle with many various lines of cooperation between the three points denoting the three separate entities: the State Department of Education, the State Board of Regents of Higher Education, and the State Department of Vocational and Technical Education. One example of joint cooperation is the shared ownership and operations of the centrally located CPU. The State Department of Education and the State Department of Vocational and Technical Education participate in this venture which is the hub of our state-wide teleprocessing educational system. Schools electing to subscribe to the system range from secondary institutions to the four-year colleges, thus the supervision of programs comes through different State agencies. With the distribution of responsibility of program supervision, there was need for additional cooperative effort. Such a need was met by funding a position to coordinate vocational education at the higher education level, and an individual was selected by and is officed with the State Board of Regents for Higher Education. As a State, as needs have arisen, attempts have been made to reach solutions. Each of these State agencies have certain and varied responsibilities and limitations when it comes to data processing education, but basically there is a framework within which progress is taking place.

The state-wide teleprocessing system previously mentioned has been in existence since 1966. It has been featured in various articles, the most recent in the January, 1971, issue of the *American Vocational Journal*. Administratively speaking the system has undergone the various growing pains of an EDP shop—hardware changes, turnover of personnel, numerous operating system modifications, and lastly a location change to a specifically designed area for the entire data center staff in a new State capital office building. Although both the State Department of Education and the State Department of Vocational and Technical Education have quite extensive administrative applications for which they use the UNIVAC 70/35, educational program use has always received top scheduling priority. Another recent improvement in the system was the conversion of TDOS to a DOS operating system. Several enhancements were incorporated that were not previously available to students. Now a student can receive a complete core-dump at a satellite center, can accomplish either tape or disk sorts, and set up the JCL for his specific job, as well as other features.

The state-wide teleprocessing system is not the only hardware available to institutions for data processing education. There are schools throughout the State on all levels that have their own computer systems. Some of these are Norman High School with a System/3; Tulsa Junior College with a 360/50; Oklahoma State University with a 360/65 (the largest central computing facility of any educational institution in Oklahoma); Oscar Rose Junior College with an IBM 1130; and Central State University with a 360/40 and two PDP 11/45's.

With all these various systems available, wouldn't it be reasonable to expect much activity? Certainly! In looking across the State, one will see scientifically-oriented programs, business-oriented programs, and an established data processing teacher education program. But is the activity well organized, and is Oklahoma really meeting the specific needs of industry for well-trained and updating of those instructors already out in the teaching profession. How does one obtain that additional training and expertise of the rapidly changing profession when he spends most of his time in the classroom or preparing a lecture to enter that classroom? Is there perhaps another avenue not yet discovered that could assist the three State agencies in working together in the midst of the intricate network of responsibilities?

Before addressing oneself to these and other specific concerns and discussing the springboard that Oklahoma is using, please allow a brief rundown of some of the usual aspects of data processing education. On both secondary and collegiate levels, the student organizations are a part of making that final employable product. FBLA (Future Business Leaders of America) and PBL (Phi Beta Lambda) provide a very meaningful and rewarding experience to students who desire to "get involved". The state winner in the FBLA data processing contest also won first place at the National Leadership Conference in Washington, D.C. last June. This young man was a high school senior and was enrolled in the data processing program at the Oklahoma City Area Vocational-Technical Center. Today he is employed in a local EDP shop and advancing very rapidly. When it comes to youth organizations and activities, Oklahoma has enjoyed an excellent romance with industry. Many businessmen serve as contest judges and ask to return for future events. Others may participate in Job Fairs, in which students are interviewed, some companies offer jobs, thus some students leave the State Leadership Conferences not only with contest awards, but also a job. Some companies desire to participate in other ways, such as presenting contest winners with savings bond certificates.

Another unusual aspect of data processing education in Oklahoma is that State level personnel are called upon to wear many different hats. Certainly this is true of the Data Center coordinator, the Data Processing Teacher Educator, and the Consultant for Secondary Programs. These three individuals speak to many civic and industrial groups, to high school classes of all kinds, and even prepare special seminar classes on various aspects of data processing for many different groups. These special seminars may be for top-level management, for a group of secretarial instructors, or for a special training session at Tinker Air Force Base near Oklahoma City. In addition, these persons work closely together in providing input concerning data processing education for the state-wide VIEW (Vital Information for Education and Work) system developed and disseminated by the State Department of Vocational and Technical Education. It would not be uncommon to find one of these persons substitute teaching (without pay) for an instructor who desires to attend a special seminar out-of-state. This instructor will in turn share the special seminar content with others via the springboard. This type of seminar, as well as one-week seminars are carefully planned and successfully implemented by these three individuals. The most recent of the week-long seminars was conducted in August, 1973. Authors and publishers from California and New York were guest lecturers. These persons, assisted by Oklahoma instructors, conducted the seminar, and presentations ranged from Techniques of Teaching Basic Data Processing Concepts to those on Techniques of Teaching Concepts of Multiprogramming, and Techniques of Teaching Concepts of Teleprocessing. These seminars alone, of course, cannot meet the total need of assisting those currently teaching data processing in keeping abreast of the state of the art and of industrial needs.

Returning now to concerns previously mentioned, other concerns have now been added to the list. What about a state-wide advisory committee from industry, and what about reducing program costs? Is there some method for obtaining some standardization of the educational product, the student, at the various levels of education? Wouldn't it be a change for education to say to industry, or more specifically for a student to enter the job market proclaiming to be an entry-level COBOL programmer and for industry to be informed specifically what criteria this student has met because of this student's participation in a competency-level testing program supported by the various institutions? Dreams? Perhaps. However, as long as it is said, "It can't be done!" it won't be done. There just may come a point in time when powers that be agree that there is a possibility for accomplishment; if so, then there is hope for progress.

This positive "system approach" has prevailed in the minds of many State level persons and in the minds of business data processing teachers in the State, and because of this and other reasons, a group of individuals are taking a stand. Statistics reveal that over 80 percent of the jobs available in the data processing arena are in the business-oriented environment. Also, there are existing vehicles to serve computer scientists and further their educational aspirations and endeavors. The National Science Foundation has funded proposals to assist the advancement of data processing in this area, and yet what about the business data processing programs and instructors? Oklahoma looked to other states and even to national organizations for some answers and assistance and found that generally the problem is the same. Business data processing education needed a boost. After much thought and careful consideration five individuals decided to create a springboard for Oklahoma. Objectives were expressed to be: (1) the continual advancement of the professional development of the business data processing instructors at all levels of education, (2) the continual improvement of business data processing programs at all levels of education, and (3) the continual promotion of business data processing. The finished product is called the Society for the Advancement of Business Data Processing Education. Does it meet the needs and objectives? Absolutely.

This product, still in its embryonic stage, is allowing the instructors from all levels of education to meet and work together, and enthusiasm and momentum continue to increase as many special projects are planned and completed. For the first time in the history of Oklahoma, a two-day seminar, specifically for business data processing instructors, was planned and well attended at the annual State teachers' meeting. This organization has voiced its support and desired involvement in developing competency-level tests for various areas of business data processing education.

Another special committee is in the process of designing an information system that will hopefully produce just about any answer one might want concerning the status of business data processing education in the State of Oklahoma. Teachers are sharing; they are involved; they want quality education program, and in all three areas value greatly the input and assistance from industry. The constitution of this newly formed group calls for state-level advisory committees, and Oklahoma is choosing such persons from industry.

Another involvement of organization members is that of writing proposals to secure special funding for a project to establish career paths for business data processing teachers, and to conduct a future two-week seminar. The group is considering supporting an attempt to establish some national curriculum standards for business data processing education at the various levels. With the close working relationship between all levels of education and the use of the springboard, these and other projects will become reality.

"Quality!" Yes, that coupled with the attitude that nothing is impossible, unless one believes it is impossible, is the cry that rises from this industrious group. They desire the involvement of industry; they want to know how they can best train workers to meet the employment demands. With the English flair of Professor Higgins, who often remarked similarly about his Dear Eliza, one may find this group of Okies cocking their heads aside, smiling, and declaring, "By George, we think we've got it!"

Career education in business data processing teacher education

by JOE M. KINZER, JR.

Central State University
Edmond, Oklahoma

The career path for teachers of business data processing has virtually been non-existent in the State of Oklahoma. Higher education institutions, however, have offered some coursework in data processing since 1958. Most of these courses were designed for scientific applications oriented toward the physicist and engineer.

Business data processing programs at the secondary, post-secondary and adult levels began in the fall of 1966 with one of the first statewide systems in the nation. At that time, prospective teachers could not be attracted from industry and special courses were designed to prepare teachers for these programs. These special courses were offered for two consecutive summers in 1966 and 1967. They included the minimum technical skill areas necessary to begin a two year or less business data processing curriculum.

Institutions offering teacher education failed to respond to the need for additional training and development of these teachers beyond the two original special courses. As a result, business data processing teachers in Oklahoma have been left entirely on their own for professional development. What has happened is that some of the teachers have taken more advanced courses in an attempt to keep themselves updated as the technology changes and some have done very little. If the level of expertise of the teacher has any effect on curriculum or on effectiveness in preparing people for career paths in the areas of business data processing, then the career development of the data processing teacher must be a high priority item by institutions of higher education, representatives of industry, professional organizations, and other interested societies.

Approximately two years ago, Central State University began an undergraduate program to train business data processing teachers. As in other states such as California, Colorado, New Hampshire, North Carolina and others, the program is offered through the School of Business as an option of Business Education. This arrangement seems to be working quite well. Courses such as Accounting, Economics, Statistics, Communication Skills and other related skill areas are taught by the School of Business while the technical coursework such as Programming Languages, Operating Systems, Systems Development and Design, and others are taught by the Department of Computer Science. The professional education skills that relate to the process of effective teaching are taught by the School of Education. Currently, plans are being formulated for a graduate program for business data processing teachers.

The area of in-service education, however, is a very high priority within the computer science education department. Oklahoma is fortunate to have a person at the State Department of Vocational and Technical Education level as a consultant in the professional development of the data processing teacher. The relationship between the State Department and the teacher education institution is excellent and allows for better planning, greater continuity, and added dimensions in in-service programs.

In summary the data processing teacher education program in Oklahoma is concentrating on three main areas of concern. The first of these is the preservice program for prospective teachers mentioned previously. The program is geared to provide the prospective teacher with the technical, related and professional skills necessary to be successful in the classroom.

The second area of concern lies in in-service education. At this time, there are no certification standards specifically for business data processing teachers. In order to keep the teacher in the classroom updated on technological changes and advancements within the industry, seminars and conferences are being used. Hardware manufacturer representatives, textbook authors, and other specialists are recruited as instructors and college credit for completion is granted, through the institution. While these seminars should be continued, an organized graduate training program for business data processing teachers is needed urgently.

The third area of concern also falls in in-service education but is structured for the secondary Business Education teacher. Many of these teachers are aware that a minimum unit of instruction (10-30 hours) should be introduced to all secondary business education students. The problem, however, is that in most cases the teacher has had little or no training in this area. Institutes and seminars are also being employed in an attempt to bridge this gap.

In talking with educators in other states and by reading some of the available literature, it appears that problems confronting the professional teachers of Business Data Processing are similar in many areas of the country. For example, should the profession demand any type of standards for the new teachers? Should these standards include industrial experience, mathematics, accounting, communication skills, and others? If so, what would be the minimum standards acceptable? Do we offer a forum such as a national organization for data processing teachers to express ideas and seek answers to their problems in the classroom? How can industry help in this effort? The list can go on and on.

Oklahoma is attempting to deal with some of these problems, but it appears that a much larger effort is needed. Guidelines for career education paths for teachers as well as for students must be defined if we are going to achieve an acceptable level of success in providing industry with a quality product.

Computer control of component insertion

by D. R. MAGILL and R. D. MEMIS

IBM Corporation
Endicott, New York

INTRODUCTION

IBM has successfully utilized computers to improve, monitor, and control actual production processes for many years and, as a result, has accumulated a large store of experience concerning their use as a manufacturing tool.

This paper describes the use of an IBM System/7 to control up to eight component insertion machines: Loose Dual Inline Package (DIP) machines and/or Variable Center Distance (VCD) machines. The application was developed and is operational on eight machines at the IBM manufacturing facility in Endicott, New York, and is now being implemented at other IBM facilities.

The DIP and VCD machines are similar in operation from the control standpoint of the System/7. The machines are multiplexed through a common interface.

The System/7 computer controls the following machine functions:

1. X, Y, Z table positioning
2. Transportation of the component through the insertion mechanism
3. Console lights
4. Sensing of console pushbuttons
5. Sensing and signaling of abnormal conditions on the machine
6. Program control of the repair of boards with missing components
7. Operator-initiated data modification
8. Component selection and control of variable center distances and insertion depths for individual components
9. Data acquisition and control (from the host system)

The component insertion machines provide a means for automatic insertion of integrated circuits. From "stick" or belt carriers, components are selected and inserted into circuit boards according to a pattern program resident in the System/7 computer memory. A machine consists of (1) control panel, (2) servo-controlled X, Y, Z axes, (3) component dispensing and insertion system, and (4) component cut and clinch unit.

INTRODUCTION TO MACHINE CONTROL

There are several ways to control machines, including:

- Wired controllers
- Programmable controllers
- Numerical control (N/C)
- Computer numerical control (CNC)
- Direct numerical control (DNC)

Each has advantages and disadvantages when evaluating for specific applications, and it is up to the user to determine what is best for his use.

Wired controllers have fixed logic and are reasonably priced with relatively low-cost components, but are not too flexible. Programmable controllers offer more flexibility, usually a slightly higher cost, but basically have limited logic and retain a sequential mode of operation. Numerical control is more expensive, quite flexible, and very repetitive, but has little overlap operation capability and has no self-correcting logic unless adaptive control is added. Computer numerical control offers more flexibility, faster debugging, and an ability to make logical changes from sensors, but costs more and is normally regarded as a standalone operation. Direct numerical control is very flexible, has facility for fast setup, debug, and turnaround, can expand to a high level of sensor control, and usually requires control of multiple devices to justify the cost, but offers the greatest opportunities for shop floor systems control.

Two types of computer control can be considered: open loop and closed loop. An open loop system has no means for comparing the output with the input for control purposes. The closed loop system has a return signal from the machine to the computer for comparative purposes.

The application of System/7 control of component insertion machines described in this paper is an example of closed loop, direct numerical control. The System/7 provides the signals to actuate the component insertion machines from preprogrammed stored programs, and also accepts feedback data that is used to provide responses according to the logic stored in the processor.

Both open loop and closed loop computer controls are used in such varied applications as milling, drilling, grinding, drafting, stamping, winding, assembling, testing, sorting, flame cutting, spraying, and in still others. Digital and analog signal processing by the System/7 affords a wide range of servo control and behind-the-tape reader (BTR) retrofit possibilities.

GENERAL SYSTEM DESIGN

System specifications and software

The System/7 communicates with another IBM computer and uses the larger host to prepare and load programs, and to receive processed sensor-based data. In such integrated systems, one or more System/7s and other processors function as a single system, distributing and interchanging data, routines, and jobs to make the most efficient use of each system's resources and abilities. A Sensor Base Control Unit (an RPQ device) between the host and the System/7 provides high-speed communication in such a multiprocessor environment.

The System/7 has four distinct components: processor module, input/output module, enclosure, and operator station. The processor module is a 5010 with 16K words of memory (16-bit word size). The input/output module is a 5012 multifunction module having two digital output groups and two digital input groups with process interrupt capability. The enclosure is C03. The operator station is the 5028.

Although a processor smaller than 16K words of memory could have been used to control the machines, a decision was made to store up to 50 part numbers in the central processing unit. The allocation of 10K words was made for programs, including MSP/7 (Modular System Programs/7) and 6K words for free space for NC data (approximately 50 part programs). (See the section entitled "Data Management".)

MSP/7 was used to program the System/7 and all operations structured around it. MSP/7 exploits the speed and responsiveness of System/7. When a high-priority interrupt occurs, MSP/7 routes the signal directly to the required routine. Programmed interrupts on the same priority level are queued and handled on a first-in, first-out basis.

MSP/7 also schedules those tasks which must be carried out at a specific time or periodically. MSP/7 allows interaction between the operator and his application. The MSP/7 library includes several groups of macros:

- Instruction macros to define the instruction set
- Specification macros to define the system configuration
- System macros to manage the system's "executive" resources
- Access macros to handle input/output facilities
- Interconnected facility macros to support multisystem communications

In this application of System/7 control of component insertion machines, MSP/7 was used for:

- Scheduling
  Timing of 10-millisecond cycle service
  Executive programs and routines
  Program timeouts
- Input/output control
- Process interrupts
- Error processing
- Operator requests

Servo control by the System/7

The VCD and DIP machines are controlled and monitored by the System/7, which is also used to close the servo loop for three axes. The physical characteristics of each axis are different, but the basic functions performed by the System/7 are the same.

The main difference is a servo data table used for each axis. The servo data table is determined by the table speed, elasticity of the system, positioning accuracy, required mass and inertia considerations, deceleration rate, etc. Having the servo data table within the System/7 program, it is very easy to alter the servo data table to attain acceptable results in the machine table.

The basic functions performed by the System/7 to control the machine table movement of each axis are to monitor the feedback counter, update the machine table's present location, calculate the delta movement required, and output the appropriate speed to the motors.

The feedback counter, used in this application, is reset each time that it is read, resulting in a displacement of the machine table from the previous reading. The feedback counter also designates whether displacement is positive or negative. The new present location is a summation of the previous present location and the feedback counter. The error factor, or Δ, is determined by subtracting the new present location from the end location. The Δ can be positive or negative, based on the direction of travel. To find the required velocity to be outputted, to compensate for the Δ, the servo data table is utilized. By means of a table lookup or by direct algorithm, the velocity can be determined and outputted.

System/7 is now controlling eight machines, each with three axes of motion, giving a total of 24 servos to be monitored and controlled. The scan rate/axis is determined by first considering counter size versus table speed, assuring that the counter will not overflow. The second consideration is the reduction of machine table overrun, which results in oscillation. For this application, with machine table speeds of 10 in./sec. and 50 in./sec., machine table overrun was the most critical. The scan rate is easily tested and modified by having this control in the System/7. The scan rate is now 10 milliseconds.

Within this 10 milliseconds time span, all 24 axes have to be serviced. This allows about 400 microseconds per axis. However, there are other machine control processes that must be controlled. These steal from the 400 microseconds. Since the axes have to be continuously monitored to maintain the positioning accuracy desired, the processing time is a major consideration. The System/7, with a 400 nanosecond cycle time, provides a good base to reduce processing time. The programming scheme is critical to the processing time required to service the axes. The most time-consuming process is the servo data table lookup.

For example, if the servo data table consists of 100 incremental speeds based on 100 different error factors, then a sequential search would require 50 executions of a search loop on the average, with the worst case being 100 executions of the search loop at the start of a long move. If a binary search is used, the average search loop is executed seven times. A better search procedure, especially applicable to servo systems, is a recursive sequential search. This search technique retains a pointer to the servo data table, based on a match of the Δ. When the Δ changes, the pointer is used as the starting point for sequential searching, and the pointer is updated when a new match is found. This method requires at most three searches before a successful match is made. The pointer is constantly changing as the table nears its end location and the Δ gets smaller.

In some cases an algorithm can be used to find the displacement in the servo table based on the error factor. This usually provides the least amount of processing and is being used in this application.

Besides the flexibility, already discussed, in designing the servo control, another major advantage of using System/7 to close the servo loop is the control of machine table limits. Based on the Δ, an operation can be initiated prior to the machine table being within its specified dead band. In this application, the insertion sequence is initiated when the machine table comes within a certain limit of the desired end location. The limits are established by weighing the mechanical delays associated with the insertion sequence against the time required to position the machine table. This greatly reduces the overall cycle time of each operation.

System control

The System/7 architecture provides program execution at four priority levels. This capability provides an effective tool for doing processing that would normally be required on a more sophisticated system. Routines which are performing rudimentary functions can be interrupted by routines required for direct process control, without any additional programming required. On this application the main uses of the priority levels are as follows:

Level 0—Basic timer interrupts
D.I. process interrupts
Level 1—Machine control routines
Level 2—Executive routine and other scheduled routines
Level 3—Initialization routines
Data management routines
Operator communication routines
Data conversion routines

The machine control functions use the basic timer to transfer control to the Executive routine every 10 milliseconds. The Executive routine executes at Level 2, passing control to the routines required to service each machine. The machine control routines execute in short bursts, at Level 1. Each routine has a series of tasks to perform, and usually needs to stay in a "loop" waiting for a machine operation to complete before going on with the next set of tasks. Each routine sets up the next routine to be executed for the machine it is now controlling. The next routine to be executed is determined by the successful or unsuccessful completion of an operation, by the machine type being controlled, or by operator action at the time. The next routine to be called is established prior to control being passed back to the Executive routine. If the machine control that is now being executed is specified as the next routine to be executed, for the same machine, then a software loop is achieved.

There are about 25 different control routines, some applicable to both VCD and DIP machines, and some unique to just one type of machine. At any one execution of the Executive routine, only one or two of the machine control routines will be executed per machine. It is possible to have all machines requiring the same routine or each requiring different routines.

Machine control

Using the System/7 to control the individual mechanical operations on the VCD and DIP machines provides increased flexibility and reduction of cycle time. In several instances, the sequence and timing of operations were modified during machine debug. The ability to alter machine operations is still being utilized, because of recommendations by manufacturing, maintenance, and engineering.

The reduction of cycle time on a machine is largely due to overlapped operations. The cycle for a DIP machine is one insertion of a component. This includes moving the part from the part station to the transfer mechanism, transferring the part to the insertion mechanism, positioning the X-Y table to the next insertion position, and performing the insert-cut-clinch operations. To achieve maximum overlap, the DIP machine was broken down into four segments. Each of these segments operates as a single unique machine which interfaces with another segment at some stage of its operation.

The four segments are:

1. Shuttle
2. Transfer
3. Insertion mechanism
4. X-Y table

The functions of the shuttle are:

1. Move the shuttle to the appropriate part station.
2. Load a DIP into the shuttle.
3. Move the shuttle to the unload position.
4. Unload a DIP into the transfer.

The functions of the transfer are:

1. Move the DIP onto the vacuum block.
2. Reset the transfer to its home position.

The functions of the insertion mechanism are:

1. Move the DIP to the insertion mechanism.
2. Insert the DIP into the board.
3. Activate the cut-clinch mechanism.
4. Reset the insertion mechanism and cut-clinch mechanism.

The functions of the X-Y table are:

1. Position the table to the X coordinate of the next insertion.
2. Position the table to the Y coordinate of the next insertion.

Considering the machine in segments enables the DIP to be moved to staging areas. This staging allows each segment to operate with the least amount of time delay due to DIP movement. The staging areas are at the points of execution, at each machine segment, where there is a dependency on another segment. At any one moment, the DIP can be staged (1) in the shuttle at the unload position, (2) in the transfer, (3) on the vacuum block, and (4) in the insertion mechanism. There is only one link from each machine segment to another which causes dependency of operation. The shuttle can operate through all its functions until the DIP has to be loaded into the transfer. At this point, the transfer has to be checked to see if it is at its "home" position and clear of DIP modules. The transfer activates as soon as it gets an indication that a DIP is in the transfer and the vacuum block is free of DIP modules. The insertion mechanism is put through its functions upon getting an indication that the DIP module is on the vacuum block. A check is made that the X-Y table is in position before the component is inserted. The X-Y table will not move until the insertion mechanism is "home."

Data management

The System/7 handles the data management for the VCD and DIP machines. Each machine has a workboard holder with one or more mounting positions for circuit cards. Tool number data maintained in the System/7 controls the positioning of the table for each of these mounting positions. Part number data, maintained in the computer, controls the positioning for insertion of components on each circuit card. In this application, each of the machines can require different tool and part numbers.

The System/7 can use four methods to acquire this data. Two of them are data links to host systems, and two are used in a standalone configuration. The first of the host data methods is the Asynchronous Communication Control Attachment (ACCA). With this attachment the System/7 simulates the performance and characteristics of the IBM 2740 Communication Terminal Model 1 with the record feature. Therefore, System/360, System/370, and the 1800 teleprocessing equipment may be used as a host system.

The other host method is the Sensor Based Control Adapter (SBCA). This provides communication with a System/360 or System/370. Both methods require MSP/7 programming support in the System/7. The SBCA method is used at this installation.

In the standalone configuration, the System/7 can receive information from either paper tape or System/7 disk. The tape reader on the IBM 5028 Operator Station reads paper tape input. The IBM 5022 Disk Storage Module acquisitions data which is stored on the disk. As in the host methods, these two methods require MSP/7 programming support in the System/7.

No matter which method is used to obtain tool or part number data, the method of handling it in the System/7 is the same. The System/7 stores data in the portion of storage unused by the control program (free space). Two directories use the first part of free space, one for tool numbers and one for part numbers. These are of a fixed length and contain directory blocks of a fixed size. When the user assembles the program, he specifies the number of directory blocks. This application uses 15 tool directory blocks and 50 part number directory blocks. The tool number and part number do not have a fixed-size lookup key. There is a maximum length for the keys; ten characters for tool number and 26 characters for part number. The keys are searched and accessed based on the number of characters typed by the operator. For example, if the part number is loaded with a key of 8521603, the operator can access this in the System/7 by just typing 85216. This method provides the flexibility required to satisfy installation of this application in several IBM locations.

The data size for both the tool and the part numbers is variable. The program dynamically partitions free space. As the program reads tool or part number data, it allocates a partition according to the size of the data.

The part number directory has several fields. These indicate the size of the allocated partition, the lookup key, where in storage the partition begins, and whether any of the machines are currently using the part number.

If the program allocates all free space or uses all of the directory blocks, the System/7 initiates automatic overlaying of data the next time the operator requests another part number. The automatic fill-in function loops through the directory until it finds a partition that is large enough for the new data and is not being used by any of the VCD or DIP machines connected to the System/7. It will then overlay the key in the directory and the data in the partition in free space.

Tool numbers do not use the automatic overlaying function. Therefore, if the tool number directory is full, or if the user requests a part number that is too large for any of the partitions, the user must clear the free space by requesting a clearing function on the IBM 5028. This function resets free space and both directories to an empty status. The user then reenters any data that the VCD or DIP machines are using and reassigns it to the machines.

When the operator assigns data to a particular VCD or DIP machine, the program verifies that the part number data is the correct type for that machine. This prevents DIP data being assigned to a VCD machine or vice versa. There is also a check made that the tool data assigned to the machine is compatible with the new part number requested.

The program allows alteration of DIP data. If a part station on the DIP machine malfunctions, the operator can request the program to skip that part station. This request alters the data until the operator assigns a new part number to that machine. At this time, the program restores the data to its original status. There is also a need to alter data while the insertion process is active. This is used to mark particular positions for automatic repair, based on operator action. The positions that are marked for repair are restored to their initial condition after the repair is processed.

Input/output processing

The machine control routines need to print messages, request data, or perform a chain of operations with multiple routings. The requirements to perform continuous machine control do not allow the control routines to handle the I/O operations. In assessing the functions required to do I/O processing, a mini-"communication network" control system was created. This control system handles a variety of devices, including the operator station, disk module, Asynchronous Communications Control Attachment (ACCA), and Sensor-Based Control Adapter (SBCA). The transactions, which occur randomly, are queued to allow for peak activity and differences in operating speeds for different I/O operations.

The system is comprised of a message table, destination table, return table, pool of buffers, and small interface routines that are device-dependent. The system is structured on the basic premise that MSP/7 macros will be used for queuing, dequeuing, processing I/O, and passing control from one routine to another. The interface routines create the standard I/O parameter list (IOLT) for MSP/7 and perform the required data conversion and reformatting.

When the machine control routines are assembled, a series of macros are used to create and structure the tables. The message table is established with each message translated to ASCII, appended with message lengths and indicators for additional information. The destination table lists each interface routine entry point with the associated interrupt level and control word to be used. This list is device-dependent and is cross-referenced by the single access macro used in the control routines. The return table is a list of return addresses to be used after I/O completion, indicated by MSP/7 routines.

The pool of buffers is actually a series of reserved storage locations chained together. These buffers are used to contain the standard IOLT for MSP/7, routing information, and the message itself. The size and number of buffers are indicated at assembly time. Each transaction processed will queue one of these buffers and create the necessary IOLT within the reserved area.

At execution time, the machine control routines and optional operator routines initiate I/O operations by the use of a single access macro, named @I/O. By coding the parameters in different ways, the tables are used to efficiently generate a code that establishes the message routing, message address, return points, and additional identifications. All I/O requests pass through a common I/O routine which allocates a buffer from the pool, forms the standard IOLT for MSP/7, and passes control to the interface routine. Upon I/O completion, the MSP/7 routine returns control to the interface routine, which determines if any further message routing is required. If not, the buffer is returned to the pool.

Display techniques for interactive text manipulation

by CHARLES H. IRBY

Stanford Research Institute
Menlo Park, California

INTRODUCTION

The Augmentation Research Center (ARC) at the Stanford Research Institute (SRI), has been developing for several years a computer-based on-line system called NLS. NLS is part of ARC's research on enhancing the intellectual effectiveness of people.[3,5,6,7,10,12,13,15] Central to the developments to date is highly interactive text manipulation using chiefly display terminals.[16,17,18] The NLS system supports a range of display terminals (from expensive text/graphics displays to inexpensive Alpha Numeric displays[1,2]) and typewriter terminals. The NLS program runs as a subsystem within a TENEX time-sharing system on a DEC PDP-10 computer.[9]

NLS is a program of about one hundred thousand instructions and about eight programmers are involved in its continued development and maintenance. Since the program has been and will be under development for several years, considerable attention is given to the employment of good software engineering practices.

NLS provides a general purpose interface to any of a large number of specialized capabilities that the user may draw upon during his work. Certain capabilities, such as text manipulation and communication with others, are important to almost any type of intellectual work, and, thus, they have received a large amount of our development resources. NLS provides the user with a consistent and coherent command language interface while allowing him to access diverse capabilities. The system is used intensely in the day-to-day work of about fifty people, some of whom access the system through the ARPA NETWORK.[11,14] These people are writers, managers, engineers, analysts, and programmers.

In addition to very flexible text editing and viewing, NLS provides the user with facilities for communication, publication-quality formatting control, numerical calculation, specialized user-supplied editing and viewing, and programming support (such as a built in debugging system and direct access to several compilers).

For a more complete description of NLS and its applications, the reader should consult References 3 and 5.

Figure 1 describes the basic structure of the NLS application program. This paper is primarily concerned with the capabilities that the Display Terminal Interface provides to the rest of the application program.

Based on the command language grammar and the user's input, the command language interpreter invokes various manipulators to modify data structures and, if appropriate, formatters to map these data structures into specified rectangular portions (called "windows") of the display screen for the user to see. User input in Figure 1 represents character input, coordinate input, and selection input (based on coordinate input).

A manipulator is that set of routines that manipulates data structures of a certain type, say type "A". An example might be the data structures used to represent a hierarchical structure that is applied to the textual information contained in the user's files. Some of the data structures are contained in the user's files; others are used to maintain user or system state information and characteristics. Such a manipulator might be applied to any of several instances of type A data structures or might always be applied to a specific instance.

A formatter consists of those routines that map a data structure of a certain type into a rectangular "window", say "a", on the display screen. Such a formatter might invoke subformatters to handle subparts of the data structure, and it might be applied to a particular instance or to any of several instances of such a data structure. A formatter might be applied to a specific window or applied to any of several windows.

In order to minimize the number of changes that will have to be made to the screen, a formatter may compare what is currently shown in the window to what is desired. To facilitate this, a formatter maintains a data structure to reflect the current contents of the window. Alternatively, the formatter may simply clear the window and format the new data into it. The size of the window (the number of characters wide and lines high) is available to a formatter from the Display Terminal Interface.

The Display Terminal Interface is that set of routines that provides the application program with primitive operations for the manipulation of and interaction with a conceptual display terminal. This interface allows the application program to support physical displays with quite different characteristics. The protocol between the terminal and

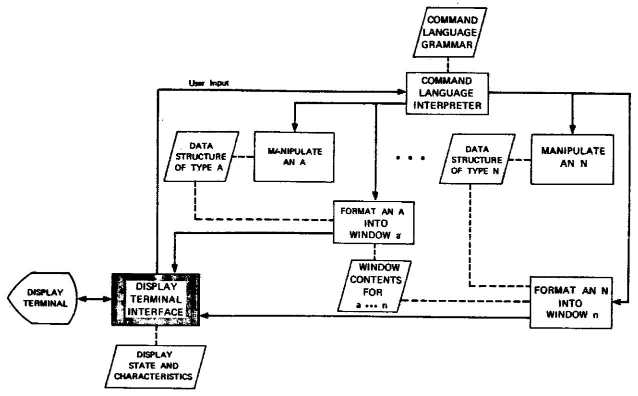

COMMAND LANGUAGE GRAMMAR
User Input
COMMAND LANGUAGE INTERPRETER
DATA STRUCTURE OF TYPE A
MANIPULATE AN A
DATA STRUCTURE OF TYPE N
MANIPULATE AN N
FORMAT AN A INTO WINDOW a
WINDOW CONTENTS FOR a ... n
FORMAT AN N INTO WINDOW n
DISPLAY TERMINAL
DISPLAY TERMINAL INTERFACE
DISPLAY STATE AND CHARACTERISTICS

Figure 1—Basic structure of NLS

the Display Terminal Interface may vary with the terminal type.

Figure 2 illustrates the window organization of a typical NLS display screen. Figure 3 shows an actual screen organized in this way. Figures 4 through 8 show other organizations on various physical display terminals.

In developing the graphics portion of the system, we wished to make use of the fairly well-known notions of "structured" display images and "virtual" display terminals in order

(1) To support a wider range of terminals without major changes to the application program.

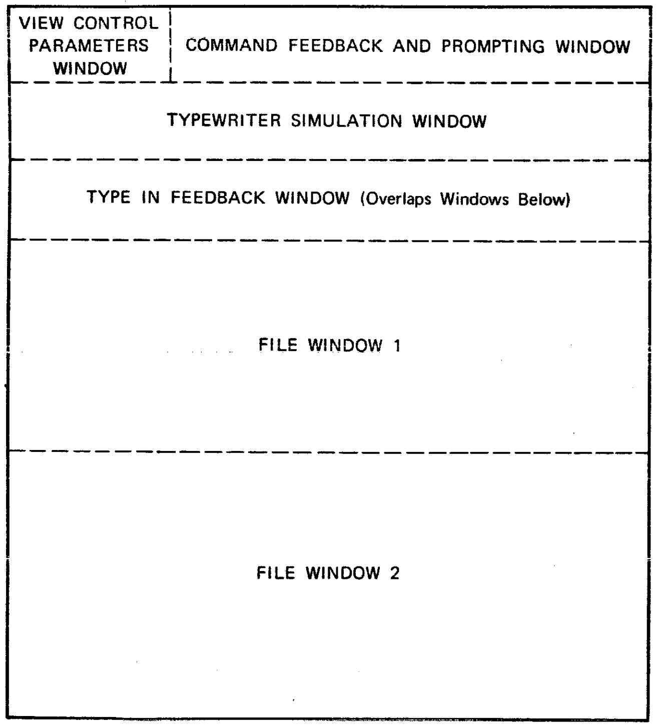
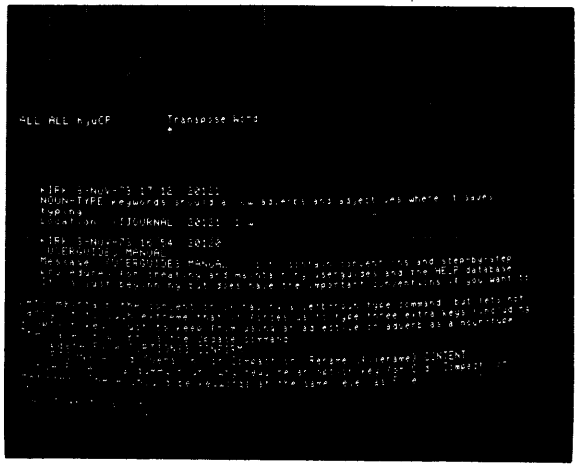

VIEW CONTROL PARAMETERS WINDOW
COMMAND FEEDBACK AND PROMPTING WINDOW
TYPEWRITER SIMULATION WINDOW
TYPE IN FEEDBACK WINDOW (Overlaps Windows Below)
FILE WINDOW 1
FILE WINDOW 2

Figure 2—A typical NLS display screen subdivided into windows—See Figure 3



```
ALL ALL [illegible]          Transpose Word

  [illegible] 5-NOV-73 17:12  20121
  NOUN-TYPE keywords should a[illegible] adverbs and adjectives where it saves
  typing
  Location: [illegible]JOURNAL 20121 [illegible]

  [illegible] 5-NOV-73 16:54  20120
  [illegible]ERGUIDES MANUAL
  Message: [illegible]ERGUIDES MANUAL [illegible] contain conventions and step-by-step
  [illegible] for creating and maintaining userguides and the HELP database
  [illegible] just beginning but does have the important conventions if you want to
[illegible]
```

Figure 3—Photograph of NLS display which corresponds to Figure 2. Typewriter simulation window and type in feedback window are empty. The display terminal is a Delta Data 5200 with a Line Processor.[1,2] Note highlighted text in upper file window, operands selected by the user for the Transpose Word command. Text may be moved from one file to another by selecting operands in separate file windows

(2) To minimize the amount of information to which a particular formatter must have access in order to modify a certain portion (window) of the display.

By "structured" display images, we mean display images subdivided into a structure (usually hierarchical or sequen-

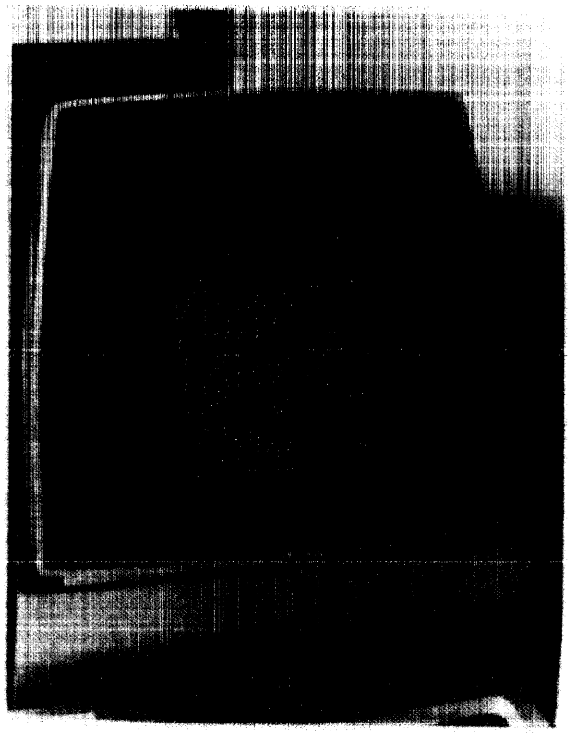


Figure 4—Photograph of IMLAC PDS-1 NLS display terminal with one file window

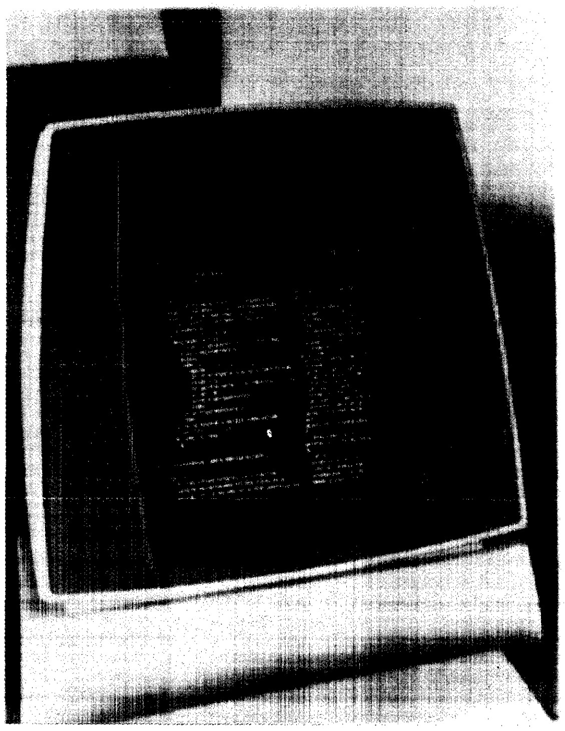
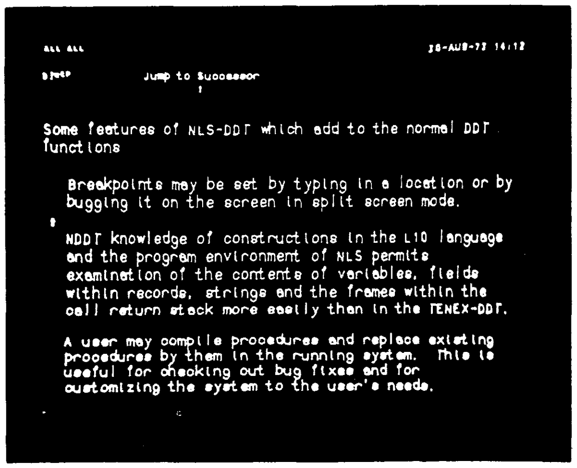


Figure 5—Photograph of IMLAC PDS-1 NLS display terminal with two columnar file windows with two different files being displayed

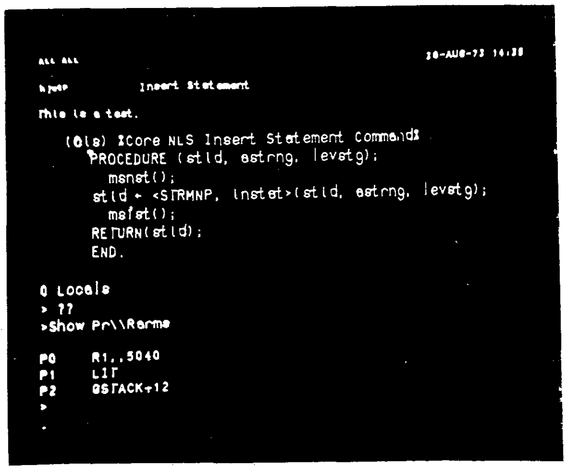
tial), such that the parts of the structure can be modified (such as deleted, moved, or replaced) independently from the rest of the display image. By "virtual" display terminal we mean a display terminal manipulated by an application program so that conceptual display properties can be mapped by interface routines into appropriate commands for the physical display being supported.

However, in attempting to apply these techniques to text display and manipulation, we discovered that there are



```
ALL ALL                                                30-AUG-73 14:12

BJMP          Jump to Successor
                         !

Some features of NLS-DDT which add to the normal DDT
functions

   Breakpoints may be set by typing in a location or by
   bugging it on the screen in split screen mode.
 !
   NDDT knowledge of constructions in the L10 language
   and the program environment of NLS permits
   examination of the contents of variables, fields
   within records, strings and the frames within the
   call return stack more easily than in the TENEX-DDT.

   A user may compile procedures and replace existing
   procedures by them in the running system.  This is
   useful for checking out bug fixes and for
   customizing the system to the user's needs.
```

Figure 6—Photograph of local display terminal screen with one file window showing documentation for debugging program



```
ALL ALL                                                30-AUG-73 14:38

AJMP          Insert Statement

This is a test.

   (Ols) %Core NLS Insert Statement Command%
     PROCEDURE (stid, astrng, levstg);
       msnst();
     stid ← <STRMNP, instst>(stid, astrng, levstg);
       msfst();
     RETURN(stid);
     END.

0 Locals
> ??
>Show Pr\\Rerms

P0    R1,,5040
P1    L1T
P2    @STACK+12
>
```

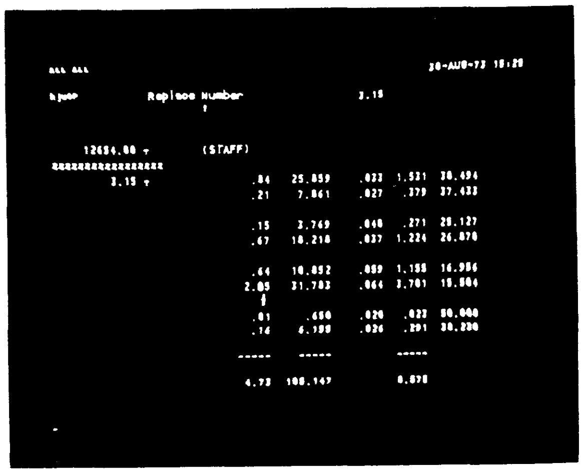
Figure 7—Photograph of display screen with one file window and one typewriter window (lower portion of screen). User may interact with NLS on upper portion of screen or with debugger on lower portion

some underlying differences between pictorial graphics, on which these techniques work quite well, and textual graphics. These differences forced us to develop a slightly different conceptual model for text displays. This paper reports what we now know about the differences and the conceptual model we have developed.

SOME FUNDAMENTAL DIFFERENCES BETWEEN TEXTUAL AND PICTORIAL GRAPHICS

Although pictorial and textual graphics are similar in most respects, there are some problems unique to textual graphics

(1) On most displays, only characters of certain sizes and spacing are acceptable to the human user (or can be displayed at all).



```
ALL ALL                                                30-AUG-73 15:20

AJMP          Replace Number                 3.15
                    !

   12654.00 +       (STAFF)
%%%%%%%%%%%%%%%%%%%
        3.15 +
                         .84   25.859    .033  1.531  30.494
                         .21    7.061    .027   .379  37.433

                         .15    3.769    .040   .271  25.127
                         .67   18.218    .037  1.224  26.870

                         .64   10.852    .059  1.155  16.956
                        2.05   31.783    .066  3.701  15.504
                          !
                         .01     .650    .020   .033  50.000
                         .16    6.199    .026   .291  30.230

                       -----    -----          -----

                        4.73  108.147          8.870
```

Figure 8—Photograph of local display screen showing use of numerical calculation features with a transaction history window on left and user's file on right

(2) Often, characters can only be displayed at certain coordinate positions with a predetermined spacing between characters. Thus, mapping a virtual coordinate system onto a physical screen may be difficult. Most displays with fixed-spaced character fonts can be thought of in terms of a character-grid coordinate system, which is not necessarily the same as its pictorial coordinate system.
(3) In general, text cannot be scaled, rotated, or translated by arbitrary amounts (as can most pictorial images).
(4) In order to control text formatting, the application program must know the character-grid coordinate system(s) of the physical display.

In order to do its job effectively (from the user's standpoint), the application program must be able to determine the usable character sizes and fonts and their associated character-grid coordinate systems for the physical display it is supporting.

These fundamental differences forced us to develop the conceptual model discussed in the following section.

A CONCEPTUAL MODEL FOR TEXT DISPLAYS

Our requirements for a conceptual model of a text display are as follows:

(1) Characteristics of the physical display should be isolated or parametric. A range of physical displays must be supported with minimal impact to the application program. We have found the "isolation of knowledge" to be an essential software engineering principle to effect long term reliability, flexibility, and maintainability of a large software system.
(2) Separate parts of the application program must be able to manipulate independently the text on portions of the screen.
(3) The user must be able to "select" text on the screen by means of some type of "pointing" device. By a pointing device, we mean a device that is capable of transmitting coordinate data to the application program (e.g., mouse,[17] stylus and tablet, joy stick) in response to some user action, such as depressing a button or a key. The pointing device should be coupled to the display in such a way that it gives the user some indication (e.g., the visual indication provided by tracking the device with a cross-hair) of what he is likely to select if he makes a selection. When the user wishes to select some text on the screen, he moves the pointing device so that it (or its displayed tracking spot) is near the desired text and takes the appropriate action to cause the coordinates to be transmitted. The application program should then determine which text is nearest the coordinates that were input and show the user what it found (e.g., by highlighting the selected text as in Figure 4). This must be done in such a way that the user can "back up" (say, by depressing some other button) and retry the selection.
(4) The ability of two or more users to "share screens" must be provided. We find great value in the ability of two or more people who are geographically distant to run NLS using display terminals through the ARPA Network and share screens. By this we mean both see the same image on their screens and both can control the application program that manipulates the image. The situation is analogous to several people standing together at a blackboard, where all can see what each writes. This sharing is greatly facilitated, of course, by a telephone connection. By this means, distributed people can work together on such things as reports, designs, papers, proposals, and computer programs. Video projectors also allow distributed meetings.
(5) If the application program needs to use the same portions of the screen for different purposes, it is very convenient for the application program to be able to suppress the display of part of the image and later to be able to restore it to sight. This is useful since most display screens are quite small, in terms of the number of readable characters they will support, and portions must often be used for several purposes. For example, the same portion of the screen might be used for the display of information from the user's files and for the display of status messages or the feedback of user input. The suppression capability allows the application program to overlap windows and use the physical screen space to best advantage without having to dedicate portions of the screen to infrequently used purposes.

   Figure 7 shows a situation where the display of a file text window has been suppressed in order for the user to interact with a debugging program in that portion of the screen. Figure 7 also shows feedback of user input (the text "this is a test") in a sequential window that extends to the bottom of the screen. As the user types more text, lines will be suppressed in the file text windows as need to avoid superpositioning.
(6) The application program must be able to draw the user's attention to some text on the screen (e.g., make it blink or increase its intensity—see Figure 3).
(7) Because of the typewriter-like interaction modes of most modern time-sharing systems, typewriter simulation should be possible on a portion of the screen when running the application program in display interaction mode (for system broadcast, error, or warning messages from the time-sharing system). We have found this to be very valuable to the user. This portion of the screen must, in general, be dedicated to this purpose because of the asynchronous nature of these messages. In Figure 7, the debugging program is interacting with the user through typewriter-

simulation on the bottom portion of the screen. The text on the upper portion of the screen is unaffected by the scrolling (simulating the behavior of Carriage Return and Line Feed) which takes place during typewriter simulation on the lower portion.

In order to meet these requirements, we have developed a conceptual model of a display terminal. The reader is referred to Figure 1, to the appendix of this paper, and to other referenced material (especially References 2 and 19) for additional details. The primary characteristics of the conceptual model are as follows.

Windows and Strings

The display screen is divisible into rectangular, possibly overlapping "windows". Windows may be invisible or visible, random or sequential. Sequential windows behave like typewriter simulations (text is scrolled though them). Random windows contain character strings which can be manipulated (moved, replaced, deleted) independently. Individual strings may or may not be selectable. Text in selectable character strings may be selected by the user via his pointing device as operands to application program commands. The terminal initially has only one sequential window that covers the whole screen and is called the "default typewriter" window.

The application program is expected to allocate windows for various types of information display to the user. Some of these windows are for the purpose of command specification feedback to the user and others are for the display of information contained in the user's files.

Basic terminal modes

The terminal can be in one of two basic modes: (1) "typewriter" mode and (2) "display" mode. In "typewriter" mode, all display windows except the default typewriter window are invisible, the default typewriter window is visible, and coordinate input is disabled; the terminal acts like an alpha-numeric display simulating a typewriter terminal. In display mode, the default typewriter window is invisible and coordinate input is enabled; the application program controls which windows are visible.

Pointing device interaction

It is assumed that in addition to character input, the terminal also transmits coordinate information along with at least certain characters. In formatting character strings that are selectable by the user (usually representing text from the user's files), the formatters construct a data structure associating each character string with the data element that it represents. When the user subsequently selects a character on the screen, the coordinates that were input are mapped by the display terminal interface, using mapping data that it maintains, into a window-identifier, string-identifier, and character count. This character and/or neighboring characters may then be "highlighted" on the screen for the user's benefit. The window-identifier, string-identifier and character count are converted by the application program, using the data structure just discussed, into data element identifiers appropriate for its use.

Sequential windows

We assume a situation where the user has only one terminal that must behave like a typewriter terminal at times and like a true two-dimensional display terminal at other times. Thus, sequential typewriter windows are very important. Any text that is received by the display that is not in the context of a display command is "scrolled" through the current typewriter window. The effect of characters like Carriage-Return and Line-Feed are simulated. We expect that, when an application program is initialized, it allocates a small sequential window somewhere on the screen and makes it the typewriter window. Thus, any error messages, system broadcast messages, terminal "linking",[9] and so forth, can be seen by the user while using the terminal in display mode.

Device specific parameters

When the Display Terminal Interface (see Figure 1) is initialized by the application program, it determines (via monitor calls or interaction with an "intelligent" terminal) enough about the display characteristics to manipulate the physical display. It returns to the application program the character-grid coordinate systems for the available character sizes of the terminal. The rest of the application program is then parameterized, on the basis of these values.

To make all of this work, we must make certain assumptions about the display (and any associated processing capability it might logically possess).

It is mandatory that:

(1) we can treat the screen like a large character grid and write characters at arbitrary positions on the grid (providing that we do not write past the edge of the screen),
(2) there is some way of mapping our conceptual display primitives, described in the Appendix of this paper, into the primitive operations of the physical display,
(3) there is some way of writing text in a mode such that it stands out from the rest (e.g. blink, reverse video, underline),
(4) there is some way of highlighting existing text on the screen in such a way that when the highlighting is removed, the original text will look just as it did before it was highlighted (This may be the same as (3) above), and
(5) there is a coordinate input device such that the cur-

rent coordinates will be input with at least certain characters and such that it can be tracked on the screen.

It is desirable but not mandatory that:

(1) there is some way of accomplishing the typewriter window capability (although this is not a must, the capability is certainly useful to the user),
(2) various (fixed spaced) fonts and character sizes are available for the terminal (we plan to extend the model to include proportionally spaced fonts in the future), and
(3) the (intelligent) display terminal is capable of responding to an interrogation command from the Display Terminal Interface. This capability is optional, since the user can supply the information instead. However, this latter approach is not very desirable or reliable.

Our operating system makes assumptions about the type of terminal that one is using. If these assumptions are incorrect (for example, if one has a display rather than a typewriter terminal), then the user must communicate this to the operating system via a command. If the terminal is intelligent and can respond to an interrogation, the user simply specifies this, and when the Display Terminal Interface is initialized, it sends the terminal an interrogation command, to which the terminal responds with its characteristics. Otherwise, the user must supply any needed information about the display terminal or its characteristics must be assumed by the application program.

THE MOUSE AND KEYSET AS IMPORTANT AIDS TO DISPLAY INTERACTION IN TEXT EDITING

Although they are very simple devices, we have found that the mouse and keyset, combined with a standard typewriter-like keyboard, form a very balanced and useful set of input devices for two-dimensional text manipulation.[4,8,16,17,18] The mouse is used for pointing and for special function input; the keyset and keyboard are used for character input. The mouse is a small device that has two perpendicularly mounted potentiometers, to which are attached wheels that roll and slide in proportion to the direction of movement, and three buttons. It is an easy "pointing" device to use and causes the user little or no fatigue. It can be used on almost any flat surface, usually a desk top.

The keyset consists of five long keys, similar in shape to white piano keys. The user depresses several keys in unison to input a character. When the thirty-one possible combinations are combined with shift buttons on the mouse, the user is able to completely duplicate the standard keyboard, while keeping one hand on the mouse, ready to point to operands for commands typed in from the keyset. When more than a few characters are to be input, the user removes his hands from the mouse and keyset and uses the typewriter-like keyboard.

The three buttons on the mouse, if depressed and released without intervening characters from the keyset, have additional functions, the more interesting of the seven being:

To select some text on the screen or give final confirmation to begin the execution of a command,

To back-up command specification, to allow the user to redo whatever he just did (e.g. select something else on the screen or retype his last character),

To abort the current command specification and return the user to the beginning of command specification, and

To allow the user to modify the parameters which control how his information is presented to him. He may do this in the middle of specifying a command.[3]

For a more extensive discussion of these devices, the reader's attention is directed to References 2 and 4.

ACKNOWLEDGMENTS

The author wishes to express his gratitude to other members of the ARC staff for their help in the development and implementation of the conceptual model discussed in this paper. In particular, he would like to thank Ken Victor for his help, criticism, much implementation work, and for maintaining the official documentation for the IMLAC protocol.[19] In addition, the author would like to thank Don (Smokey) Wallace for implementation help, Don Andrews for his work on the Line-Processor,[2] and L. Peter Deutsch, of XEROX Palo Alto Research Center, for helping to develop the protocol for supporting IMLAC PDS-1 display terminals and for writing an IMLAC program to implement it.

The work reported here was and is currently being supported primarily by The Advanced Research Projects Agency (ARPA) of the Department of Defense, and also by the Rome Air Development Center of the Air Force and the Office of Naval Research.

REFERENCES

1. Hardy, M. E., "Micro Processor Technology in the Design of Terminal Systems," under Preparation for the *Proceedings of the IEEE COMPCON*, 1974, SRI-ARC Catalog Item 20185.
2. Andrews, D. I., "Line Processor: A Device for Amplification of Display Terminal Capabilities for Text Manipulation," prepared for the *Proceedings of the National Computer Conference*, May 1974, SRI-ARC Catalog Item 20184.
3. Engelbart, D. C., R. W. Watson and J. C. Norton, "The Augmented Knowledge Workshop," *AFIPS Proceedings National Computer Conference*, June 1973, 38 p., SRI-ARC Catalog Item 14724.
4. Engelbart, D. C., "Design Considerations for Knowledge Workshop Terminals," *AFIPS Proceedings National Computer Conference*, June 1973, 38 p., SRI-ARC Catalog Item 14851.
5. Engelbart, D. C., *SRI-ARC Summary for IPT Contractor Meeting*, San Diego, 8-10 January 1973, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, 7 January 1973. 8 p., SRI-ARC Catalog Item 13537.
6. Engelbart, D. C., *Coordinated Information Services for a Discipline- or Mission-Oriented Community*, Stanford Research Institute, Aug-

mentation Research Center, Menlo Park, California, paper given at Second Annual Computer Communications Conference, San Jose, California, 24 January 1973, 12 December 1972, preprint, 13 p., SRI-ARC Catalog Item 12445.

7. Augmentation Research Center Staff, *Online Team Environment: Network Information Center and Computer Augmented Team Interaction*, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, RADC-TR-72-232, 8 June 1972, 266 p., SRI-ARC Catalog Item 13041.
8. Savoie, Robert, *Summary of Results of Five-Finger Keyset Training Experiment*, Project 8457-21, Stanford Research Institute, Bioengineering Group, Menlo Park, California, 29 March 1972, 4 p., SRI-ARC Catalog Item 11101.
9. Bobrow, D. G., J. D. Burchfiel, D. L. Murphy and R. S. Tomlinson, "TENEX, A Paged Time Sharing System for the PDP-10," presented at *ACM Symposium on Operating Systems Principles*, 18-20 October 1971, Bolt Beranek and Newman Inc., 15 August 1971, SRI-ARC Catalog Item 7736.
10. Engelbart, D. C., *Network Information Center and Computer-Augmented Team Interaction*, Interim Technical Report, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, RADC-TR-71-175, AD 737 131, July 1971, 104 p., SRI-ARC Catalog Item 8277.
11. Roberts, L. G. and B. D. Wessler, *The ARPA Network*, Advanced Research Projects Agency, Information Processing Techniques, Washington, D. C., May 1971, SRI-ARC Catalog Item 7750.
12. Engelbart, D. C., *Experimental Development of a Small Computer-Augmented Information System*, Annual Report, 15 April 1970, 15 April 1971, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, 15 April 1971, 8 p., SRI-ARC Catalog Item 8616.
13. Engelbart, D. C. and Staff of ARC, *Advanced Intellect-Augmentation Techniques*, Final Report, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, CR-1827, July 1970, 212 p., SRI-ARC Catalog Item 5140.
14. Engelbart, D. C., "Intellectual Implications of Multi-Access Computer Networks," paper presented at *Interdisciplinary Conference on Multi-Access Computer Networks*, Austin, Texas, April 1970, Preprint, 12 p., SRI-ARC Catalog Item 5255.
15. Engelbart, D. C. and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, Vol. 33, 1968, 15 p., SRI-ARC Catalog Item 3954.
16. Engelbart, D. C., W. K. English and J. F. Rulifson, *Development of a Multidisplay, Time-Shared Computer Facility and Computer-Augmented Management-System Research*, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, AD 843 577, April 1968, 180 p., SRI-ARC Catalog Item 9697.
17. English, W. K., D. C. Engelbart and M. A. Berman, "Display-Selection Techniques for Text Manipulation," in *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, No. 1, March 1967, p. 5-15, SRI-ARC Catalog Item 9694.
18. English, W. K., D. C. Engelbart and Bonnie Huddart, *Computer Aided Display Control*, Final Report, Stanford Research Institute, Augmentation Research Center, Menlo Park, California, CR-66111, N66-30204, July 1965, 104 p., SRI-ARC Catalog Item 9692.
19. Staff of ARC, *IMLAC User's Guide*, Stanford Research Institute, Augmentation Research Center, unpublished, available upon request.

APPENDIX—PRIMITIVES OF THE CONCEPTUAL MODEL OF A TEXT DISPLAY

The primitive operations that the Display Terminal Interface provides to the application program are listed here.

For window Manipulation:

WINDOW-ID ← ALLOCATE-WINDOW (X1, Y1, X2, Y2, CHARACTER-SIZE, FONT, TYPE)

Function: Allocates a rectangular window of the specified type (random or sequential) and position. Establishes default character size and font for the window.

Arguments:

X1, Y1: screen coordinates of upper left corner of window.

X2, Y2: screen coordinates of lower right corner of window.

CHARACTER-SIZE: default character size for this window.

FONT: default font for this window.

TYPE: sequential or random

Returns:

WINDOW-ID: unique identifier for this window (to be used in subsequent commands).

DEALLOCATE-WINDOW (WINDOW-ID)

Function: Deallocates the specified window.

CLEAR-WINDOW (WINDOW-ID)

Function: Deletes contents of window and removes image from the screen.

INVISIBLE-WINDOW (WINDOW-ID)

Function: Makes the contents of the window invisible (no image on the screen).

VISIBLE-WINDOW (WINDOW-ID)

Function: Makes the contents of the window visible (image appears on the screen).

TYPEWRITER-WINDOW (WINDOW-ID)

Function: Makes the specified (sequential) window the typewriter window. All "unescorted" characters (not within a display command) will be scrolled through this window. These characters can also be scrolled through the default typewriter window so that the user can see them when the terminal is returned to typewriter mode.

For Character String Manipulation:

STRING-ID ← WRITE-STRING (WINDOW-ID, X, Y, CHARACTER-SIZE, FONT, HIGHLIGHT, SELECTABLE, CHARACTERS)

Function: write the specified string in the window, with the specified properties at the specified position.

Arguments:

WINDOW-ID: unique identifier for a window.

X, Y: window coordinates of the first character of the string.

CHARACTER-SIZE: Use specified character size for this string or use window default.

FONT: Use specified font for this string or use window default.

HIGHLIGHT: If specified, highlight this string (make it stand out to user).

SELECTABLE: If specified, characters in this string may be selected by the user via the SELECT-CHARACTER primitive.

CHARACTERS: the characters to be displayed.

Returns:
STRING-ID: unique identifier for the string within this window.

REPLACE-STRING (WINDOW-ID, STRING-ID, X, Y, CHARACTER-SIZE, FONT HIGHLIGHT, SELECTABLE, CHARACTERS)
Function: Replaces the specified string in the specified window by the characters specified. If the X, Y coordinates are not specified, the current position is used. FONT and CHARACTER-SIZE may be defaulted to the old values for the string or to the window defaults. If HIGHLIGHT is specified, the string is made to stand out from normal text on the screen.

MOVE-STRING (WINDOW-ID, STRING-ID, X, Y, CHARACTER-SIZE, FONT, HIGHLIGHT, SELECTABLE)
Function: Move the specified string to the specified position within the window.

DELETE-STRING (WINDOW-ID, STRING-ID)
Function: Delete the specified string from the specified window.

INVISIBLE-STRING (WINDOW-ID, STRING-ID)
Function: Make the specified string invisible to the user (no image on the screen).

VISIBLE-STRING (WINDOW-ID, STRING-ID)
Function: Make the specified string visible to the user (image on the screen).

CLEAR-STRING (WINDOW-ID, STRIND-ID)
Function: Same as REPLACE-STRING with null string.

For Sequential Window Manipulation:

APPEND-TEXT (WINDOW-ID, CHARACTERS)
Function: Append the specified characters to the specified sequential window. Carriage Return and Line Feed characters are simulated within primitive and are automatically inserted to avoid characters exceeding the right edge of the window.

For Highlighting Characters:

MARK-CHARACTERS (WINDOW-ID, X1, X2, Y)
Function: Highlight the characters from position X1, Y to X2,Y in the specified window, such that the mark can be removed with REMOVE-MARK and the original characters will be unchanged. It is desirable, but not mandatory, for the user to be able to read characters that are marked by this primitive.

REMOVE-MARK ()
Function: Remove the last mark put on the screen with MARK-CHARACTERS.

CLEAR-MARKS ()
Function: Remove all marks put on the screen with MARK-CHARACTERS.

For Cursor Manipulation:

SET-CURSOR (CHARACTERS)
Function: If possible for this display terminal, set the primary cursor (the one that tracks the user's pointing device) to the specified characters.

PLOT-SECONDARY-CURSOR (X,Y, CHARACTERS)
Function: Plot a secondary cursor at screen position X,Y using the characters specified if possible. This must be done in such a way that the original text on the screen is not destroyed. This primitive is used in screen sharing.

For User Input:

CHARACTER ← READ-CHARACTER ()
Function: Read the next character input from the terminal.

(X,Y) ← READ-CURSOR-COORDINATES ()
Function: Read the next (screen) coordinates iuput from the terminal.

SEND-COORDS-WITH-CHARACTERS ()
Function: Begin sending cursor (screen) coordinates with (at least certain control) characters.

DONT-SEND-COORDS-WITH-CHARACTERS ()
Function: Stop sending cursor coordinates with any characters.

TIME-INTERVAL-COORD-INPUT (TIME-INTERVAL)
Function: Begin reporting cursor coordinates periodically (when they have changed), independent of user actions, for use in screen sharing.

For User Selection of Text on the Screen

(WINDOW-ID, STRING-ID, CHARACTER-COUNT, X′, Y′) ← SELECT-CHARACTER (X,Y)
Function: Given the screen coordinates X,Y, find the nearest selectable character on the screen.
Returns:
WINDOW-ID: The unique identifier for the window containing the string that contained the selected character.
STRING-ID: The unique identifier for the string containing the selected character.
CHARACTER-COUNT: The index into the string identified by STRING-ID of the character that was selected.
X′, Y′: The window coordinates of the selected character.

WINDOW-ID ← SELECT-WINDOW (X,Y)
Function: Given the coordinates X,Y, return an identifier for the nearest window containing selectable character strings. Such windows should not overlap.

For Batch Processing Display Commands:

PROCESS-COMMANDS (DISPLAY-COMMANDS-LIST, WINDOW-ID)
Function: Given a list of display commands (like those described above), perform the operations all at once on the display in a manner appropriate to the actual display.

For Error Messages:

OUTPUT-ERROR-STRING (CHARACTERS)
Function: Output the error message in a manner appropriate to the display.

For Determining Display Characteristics:

PARAMETERS ← INTERROGATE-DISPLAY ()
Function: Determine the usable character sizes, fonts, and character-grid coordinate systems for the display.

The "normal" character size and font are also indicated. Execution of this primitive also initializes the Display Terminal Interface routines to work with the actual display.

For Basic Mode Switching:

TYPEWRITER-MODE ()

Function: Put the terminal in typewriter mode. Make all windows invisible except for the default typewriter window and disable coordinate input.

DISPLAY-MODE ()

Function: Restore terminal to display mode. Make default typewriter window invisible, make any windows that were visible prior to the last TYPEWRITER-MODE command visible again, and enable coordinate input.

For Resetting the Terminal to Its Initial State:

RESET ()

Function: Reset the display terminal to its initial state, simulating a typewriter-like terminal with no windows allocated and not sending coordinates with any characters.

Line processor—A device for amplification of display terminal capabilities for text manipulation

by DONALD I. ANDREWS

Stanford Research Institute
Menlo Park, California

INTRODUCTION

The Line Processor is a microcomputer-based device that was developed at Stanford Research Institute's Augmentation Research Center (ARC). It was designed to make it possible to use inexpensive alphanumeric video display terminals with ARC's sophisticated interactive information manipulation system, NLS.[1]

NLS is a software system that runs in a timesharing environment and is accessed via enhanced display terminals we call workstations, in many cases through a computer network.

The Line Processor concept involves placing a processor in the transmission line between the main computer and the display terminal. This enables us to expand the capabilities of the alphanumeric display terminal so that it will meet the requirements of an NLS workstation terminal. The contributions of the Line Processor concept for the most part address the needs of the software system NLS and are as follows:

The microcomputer implements a "virtual two dimensional alphanumeric terminal." Primarily, the virtual terminal allows device independent manipulation of displayed text. The microprocessor "maps" the virtual terminal commands from the main computer into actual commands for the particular display attached to it.

The "virtual terminal" was designed to mate with the interactive text manipulation techniques developed at ARC especially for NLS. Understanding these techniques is most useful in understanding the Line Processor operation.[2]

Attached directly to the Line Processor is a pointing device that allows the user to point to any character on the screen. This is achieved without any modifications to the standard alphanumeric video display terminal.

The Line Processor also implements a "scrolling window" feature. This makes the terminal usable as both a two-dimensional applications display terminal and a teletypewriter terminal simultaneously, without scrambling of text. The "scrolling window" concept is extremely important in an application such as ours where the display terminal is used for both purposes simultaneously.

Communication to and from the main computer consists of 7-bit ASCII characters over bit-serial transmission lines or computer networks.

In developing the Line Processor, we have established what we feel is the minimum set of display terminal capabilities that enable two-dimensional interactive text manipulation. These requirements are discussed in detail.

Before explaining the Line Processor characteristics and operation, we will briefly describe the NLS software system and the NLS workstation environment.

NLS AND THE WORKSTATION ENVIRONMENT

NLS is a highly interactive system that joins many capabilities. It supports structured text files, very flexible editing, techniques for viewing and studying, arbitrary word processing and document production, and aids to applications and system programming. (For a general description of ARC's goals and efforts see Reference 1 which sites related papers.)

The system can be operated from a range of terminal types: from typewriter terminals to high speed graphics displays. Our main effort has been to develop a carefully human engineered display oriented system. The display version of the system introduces the user to the world of two-dimensional computer interaction, which is much more natural than the one-dimensional mode forced upon typewriter device users.[2]

The NLS program runs as a subsystem of a TENEX timesharing system on ARC's Digital Equipment Corporation PDP-10. The PDP-10 is connected to the ARPA network and many of our users gain access via that network.[3, 4]

NLS users employ two special input devices continuously: a mouse and a five-finger keyset. We use the term "workstation" to mean a display with standard keyboard and these devices, arranged on a special table in a convenient manner for effective working. (See Figure 1.)

The two unusual workstation input devices, the mouse and keyset, are commercially available separately, but are offered as standard options on only a few products. (Mice and keysets can be purchased from Imlac, Cybernex and Computer Displays.)

The five-finger keyset has five long keys as shown in Figure 2. The user rests his fingers lightly on the keys and strikes chords to input characters. The typical user learns enough binary codes in a couple of hours to do useful work.

The user can point to characters on the screen by rolling

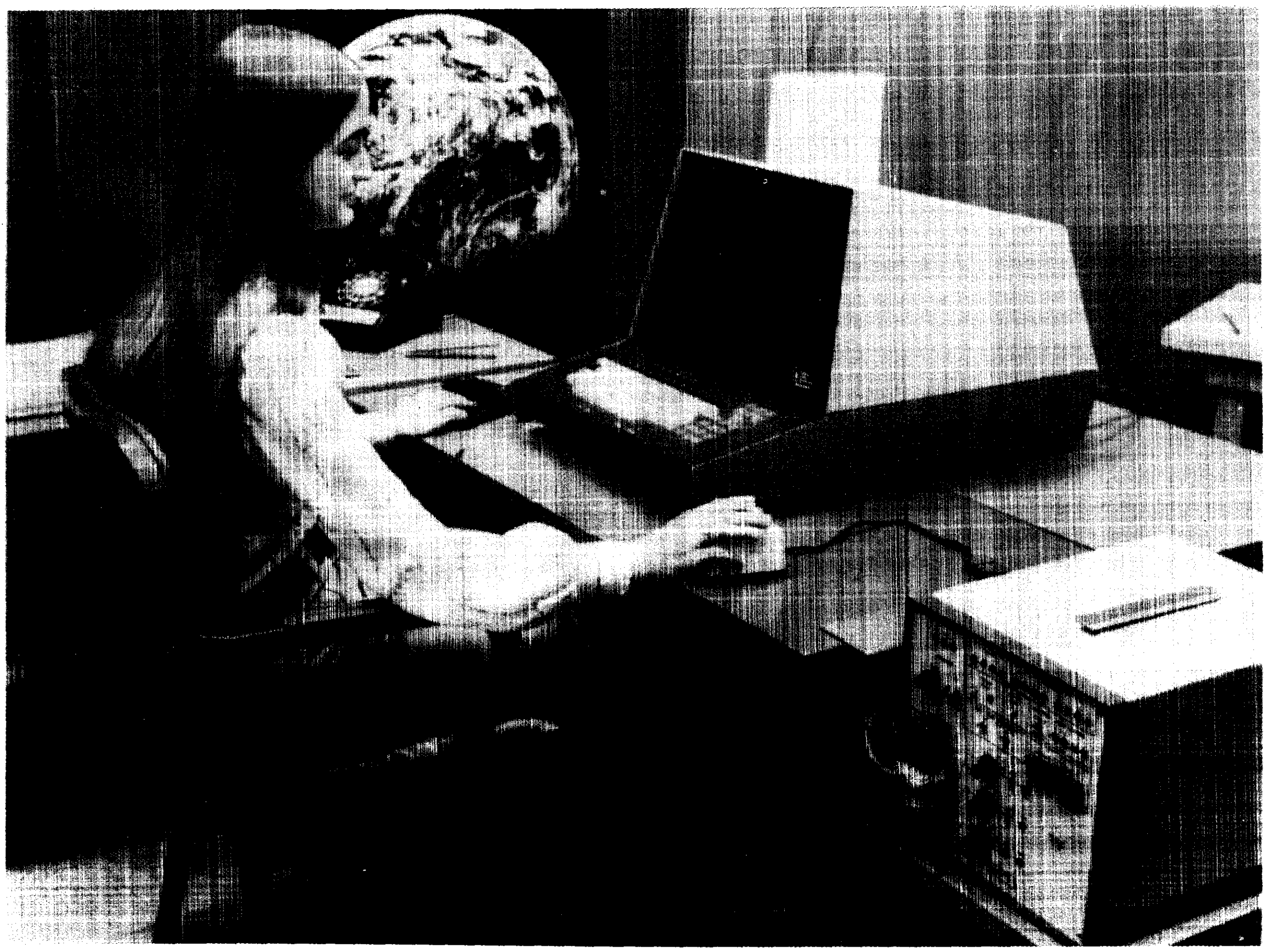


Figure 1—NLS workstation with line processor

the mouse on any flat surface. Potentiometers are connected to the orthogonal wheels on the mouse, and they provide analogue signals that are translated into digital X and Y position coordinates by a two channel A to D convertor. (See Figure 3.)

The user can operate three buttons on the mouse to confirm commands, abort commands, provide a case shift for the keyset and perform other programmable functions. The workstation has no other special function keys or buttons.

With the right hand on the mouse, and the left hand on the keyset, the user can edit, change his view and perform other control operations rapidly and easily. He does not remove his hands from the mouse and keyset, except to type large amounts of text. The mouse and keyset are further described in References 5 and 6.

The final element of the workstation environment is the user. It is intended that the user does the majority of his day-to-day work via NLS and the workstation. The user becomes increasingly proficient at using the system and the human engineering aspects of the workstation become very important.

GOALS AND PROBLEMS

For several years, we have been wanting to make NLS available to remote users on inexpensive display terminals. The primary problem has been lack of commercial availability of adequate displays. At ARC, we have a custom-built display system that meets our needs, but is not available to others.[7]

To be an adequate NLS workstation, a display terminal must meet several general requirements such as screen size and character set. These will be discussed later.

In addition, we require that the terminals have a mouse or comparable pointing device.

Also we prefer to use terminals that are commercially available and maintained nationwide.

Remote operation over high speed phone lines and computer networks such as the ARPA Network is absolutely necessary.

Further, they should be available in single quantities, and operate as a time-sharing system typewriter terminal as well as an NLS workstation.

Very few displays on the market meet our requirements. At this time, the only commercially available product with mouse and keyset that meets our requirements is a minicomputer-based display terminal, the IMLAC PDS-1. It makes a satisfactory NLS workstation, but appropriately configured it costs about $18,000. We would like to see workstations available for about $5,000.

Many low-cost, alphanumeric video display terminals on the market today are attractive and almost meet our needs. Unfortunately, many are designed as replacements for typewriter terminals, IBM 2265's, or other existing terminals. From our point of view, the designers of these terminals took an unnecessarily limited view of their possible applications.

Even those alphanumeric video terminals that meet our general requirements are not suitable for NLS use since they lack provisions for adequate graphical pointing devices—and it is generally very expensive or impossible to interface devices such as a mouse to existing products.

In addition, we would like our users to have a choice of terminals to use as workstations. But there is no industry standard for the terminal function codes, such as delete line. What is worse, there is no general standard as to how some of the functions are carried out on the screen. Hence we have to write and maintain a software driver for each different type of terminal we support. We already support a number of different hardcopy terminals, and we would prefer not to aggravate the problem if possible. In our development, we rewrite, modify and extend large parts of our software system continually and increasing the volume and complexity of the software naturally increases our development and maintenance expense.

SOLUTION

To solve the problems mentioned above, we built a special purpose I/O device based on a four-bit microcomputer, which we call a Line Processor.

The combination of a Line Processor and an adequate video display terminal results in an effective NLS workstation.

The Line Processor connects to the display terminal's RS232 bit-serial interface. No modifications are required of the display terminal. (See Figure 4.)

The mouse and keyset connect directly to the Line Processor. All special purpose hardware for these connections is neatly localized within the Line Processor.

Another standard RS232 full-duplex interface connects the Line Processor and the main computer. Communications over these lines consists of 7-bit ASCII characters. ASCII control characters are avoided to the extent that transmission over the ARPA Network involves a simple TTY-type connection.

The microcomputer implements the "virtual NLS terminal." It maps the terminal-independent Display Manipulation Protocol into the specific terminal functions for the particular terminal being used. All composite terminals (display terminal plus Line Processor) are logically the same

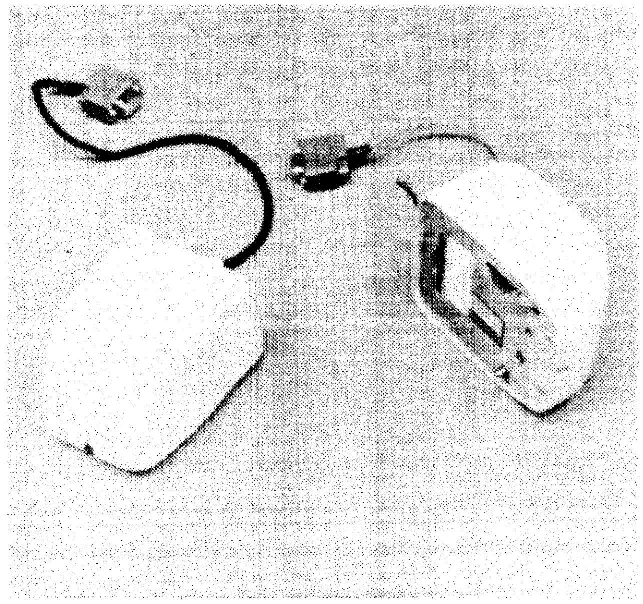


Figure 2—Mouse

at the point of main computer connection, and as a result, only one software driver is required.

The microcomputer can be programmed to work with any alphanumeric display that meets certain requirements described below.

The microcomputer program does not require any software maintenance expense, since it is treated as a hardware device when the development is completed. On the other hand, the microcomputer program could be changed without altering the Line Processor hardware, if that proved to be useful.

REJECTED SOLUTIONS

We considered several alternative solutions to the problems mentioned above. Here are the other alternatives that we investigated:

1. Have a special workstation terminal built by a reputable manufacturer. In 1972, we sent out a request to several manufacturers for a proposal to modify their standard terminal to meet our requirements. There was virtually no interest in serving our needs, and all indications were that special workstation terminals would be very expensive. The manufacturers either were not interested in modifying their standard products or would not address themselves to what they viewed as a relatively small market.

   We expect that in the future video terminals will have the necessary I/O capabilities and be microcomputer controlled so that adaptation to applications such as ours would be relatively easy.
2. Modify an existing alphanumeric terminal ourselves. There are several disadvantages to this approach. Having the modifications made on a production basis

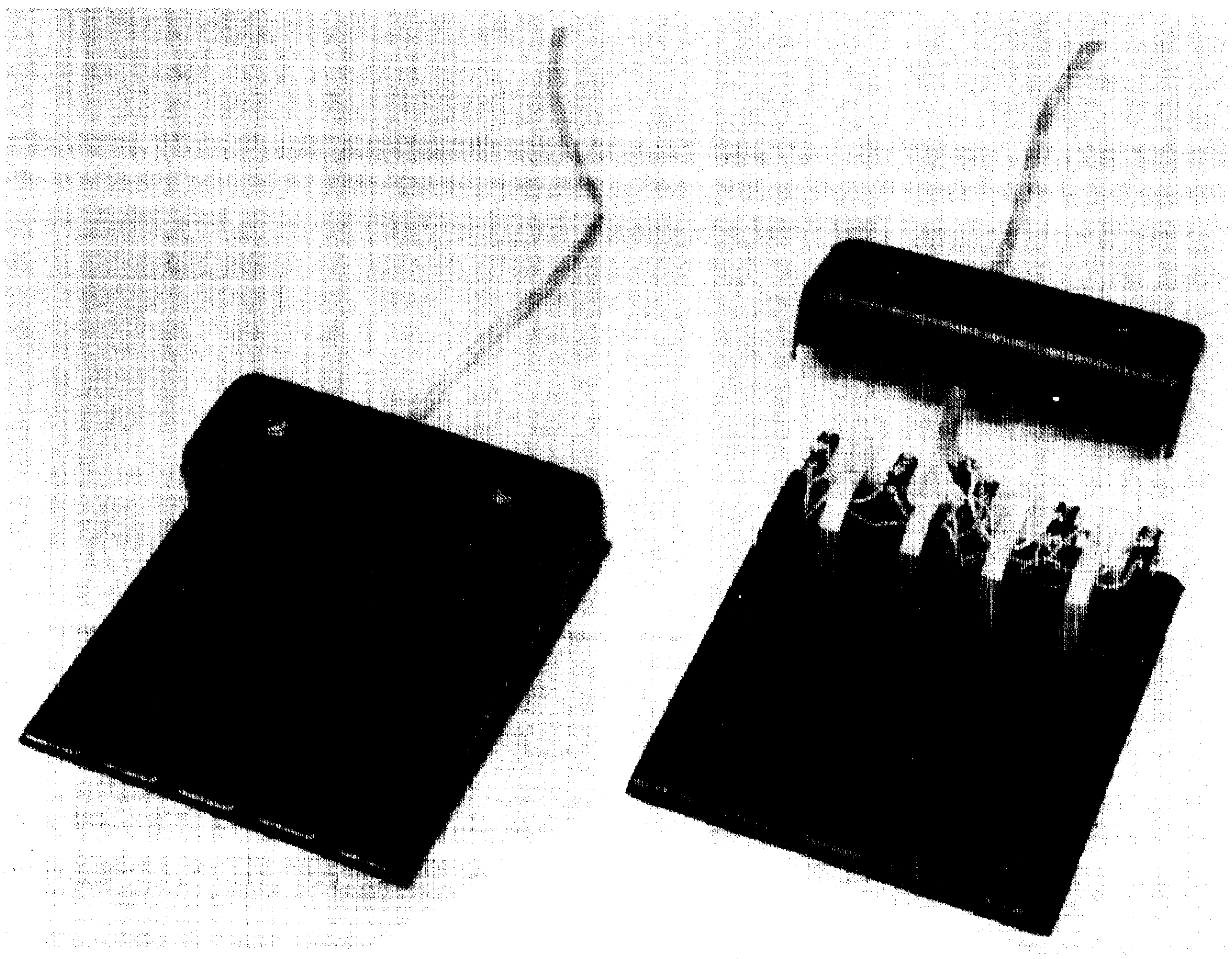


Figure 3—Keyset

and convincing the manufacturer to maintain modified terminals are big obstacles. Also, once the terminal is obsolete or no longer manufactured for any reason, we are faced with our original problems again.

3. Use of a programmable terminal that offers the necessary I/O capabilities. This is exactly what we did by programming the IMLAC PDS-1, but we feel that it results in a workstation that is too expensive. Other programmable terminals that met our general requirements exist in the same general price range as the PDS-1; one of our primary objectives was to reduce the price per terminal. But the price of suitable programmable terminals may drop enough in the next few years that this will be an attractive solution.
4. Build a very simple hardware device to interface the mouse and keyset to a communications line. The Line Processor performs several functions that need not be performed at the terminal site. For example, mouse tracking, TTY-simulation and display manipulation protocol implementation could be performed in the main computer. The device proposed here would simply transmit keyset chords and mouse position changes to the main computer.

There are two serious problems with this solution. Tracking the mouse by the main computer, when connected by a computer network, would be unfeasible because of network delay times. Also, the terminal would not have the "scrolling window" feature.

This is a reasonable idea but, compared to the Line Processor, the device would increase significantly the compute load on the main computer, and it would increase the amount of transmission over the terminal-computer connection. Furthermore, it would require more software in the main computer. In a large and fixed system where terminals were not connected by networks, this may be an advisable approach, but we felt that it was unsuitable for our configuration. Our software system is continually evolving, our main computer is overloaded and we have many of the terminals connected via computer networks.

In considering each of these alternatives, we were concerned with the cost of the development, but primarily we were concerned with the quality of the resulting NLS workstation and the end cost to the consumer. Providing a means of access to NLS for remote users is one of many R&D goals at ARC. Hence the cost of development of the Line Processor was borne by ARC and its sponsors, and did not have to be passed on to the consumer.

LINE PROCESSOR FUNCTIONS

There are several important functions that the microcomputer performs to make the combination of display and Line Processor an effective workstation.

1. Protocol Implementation—There are several protocols involved. They are identified in Figure 4.

The Display Manipulation protocol is exactly the same for every Line Processor workstation and is sent by the applications program to the Line Processor to change the display image. It does not affect the display terminal directly, but is translated by the microcomputer into the Terminal Function protocol.

The Display Manipulation protocol is designed to work with any alphanumeric terminal with cursor control and line editing functions such as delete line and insert line.

The Line Processor talks to the display terminal in the Terminal Function protocol. This is defined by the terminal manufacturer and usually consists of ASCII control codes, or sequences of control codes, interspersed with ASCII text to be written on the display screen.

The Line Processor workstation serves as both a timesharing system typewriter terminal and a two-dimensional applications system display output terminal. Hence, there are potentially two streams of output going from the main computer to the Line Processor on the same communication line: the Display Manipulation protocol, and any teletypewriter terminal output that the timesharing system or applications programs send. The teletypewriter output would be generated if the user were using the terminal as a typewriter terminal, or if the user received an error message or some type of system-wide message. These two streams of output are separated by the Line processor, and TTY-type output is displayed in a TTY-simulation area. This means that the teletypewriter output is not scrambled in with the display output, but it is scrolled—teletypewriter fashion—in a small portion of the screen. The applications program has control over the size and location of the TTY-simulation area. (The TTY-simulation and window concepts are described in reference 2.)

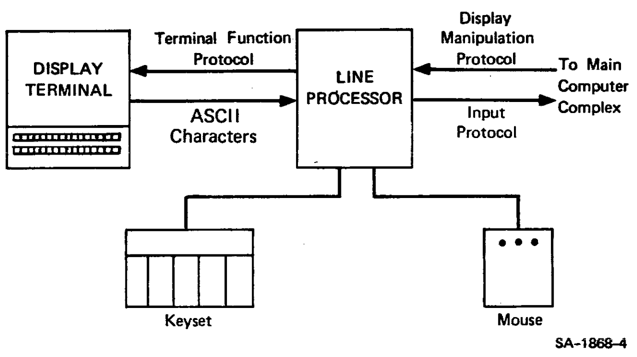


Figure 4—Display terminal, line processor and main computer connections

2. Keyset and Mouse Button Transmission (Input Protocol)

Information from the input devices (keyboard, mouse and keyset) are incorporated into an Input protocol by the Line Processor and sent to the main computer. The protocol is quite simple. Control characters and mouse button changes are sent as short strings of characters that include the mouse tracking spot location (line and character position). Characters from the display's keyboard are sent in unescorted fashion, that is, each single ASCII character is sent.

Input from the display terminal's keyboard simply comes into the Line Processor from the full-duplex display connection. This consists of ASCII characters that the user has typed on the keyboard. Local editing from the keyboard and batch transmission features within the terminal, if present, are not used for NLS applications.

There are three buttons on the mouse. A change of button positions sends in one character to the main computer, which indicates the new three-button status. The applications program knows the state of the mouse buttons and can interpret any subsequent input accordingly. This in effect allows eight different interpretations of keyset and keyboard input.[2]

The five-finger keyset input is interpreted as "chords". The normal condition is all-keys-up, and the end of a stroke is indicated by all-keys-up. Any keys that were depressed during the stroke are "OR'ed" together to comprise the chord, which eliminates the problem of striking keys in unison. Thus, a keyset input stroke is a non-zero, five-bit integer. It is converted to an input character in the Line Processor, and sent on to the main computer in unescorted fashion.

The five-finger keyset and mouse buttons are gated directly into the microcomputer. These signals are not "clean" and may be "bouncing" for a few milliseconds after a button is pushed. The switches are sampled at a rate at which the bouncing can be detected, and the final reading is not taken until the switches are stable.

3. Mouse Tracking

The Line Processor reads the mouse position from the A to D convertor and "tracks" its movements on the screen with the standard display cursor implemented within the video terminal. This is done by periodically reading the mouse position and, if it has changed, sending a position-cursor command (a part of the Terminal Function protocol) to the display.

The tracking allows the user to point to any character on the screen at any time, yet the main computer is only informed of the mouse position when the user strikes an appropriate key or button.

The mouse moves in a smooth continuous manner, but the tracking spot (cursor) moves from one character position to the next in discrete jumps. The tracking will appear jumpy rather than smooth if the baud rate between the Line Processor and the display is not high enough. Exactly what the baud rate has to be depends on how many characters must be sent for the position cursor function. We have found that 2400 baud is a minimum baud rate with the standard four-character sequence for cursor positioning found on most alphanumeric terminals.

4. Display Control

The Line Processor maintains control of the display because it is the only device sending characters to the display terminal. The applications program in the main computer manipulates the display only by way of the Display Manipulation protocol.

The mouse tracking and the TTY-simulation feature mean that the microcomputer must have a fair amount of software logic to keep track of the display terminal's cursor position and know where to move it when necessary.

TERMINAL REQUIREMENTS

A display terminal must meet several requirements to form an effective workstation with a Line Processor. These requirements are due to either system or user considerations:

1. System Considerations

It must be possible to perform several kinds of display functions by way of the terminal's RS232 connection. That is, the Terminal Function protocol must be adequate and operable under computer control. We will discuss the actual functions that are required.

One of the key functions is positioning the cursor. For the mouse tracking to be satisfactory, it must be a very quick process (within one millisecond), and the cursor must not be displayed in any extraneous positions on the screen while being moved to the destination. The cursor must be addressed by the character and line number of the screen position.

The appearance of the cursor must be suitable for tracking. A blinking underline cursor is not very satisfactory since it is not always visible. Some displays implement the cursor by reversing the video in the entire dot matrix of the character in question. The resulting reverse video rectangle that moves around the screen is satisfactory for tracking.

It is necessary to have a high speed connection between the Line Processor and the display, as mentioned before. We feel that 2400 baud is satisfactory; we prefer 9600 baud because it makes a more effective workstation.

The bulk of the display manipulations are done via delete line and insert line functions. These manipulations usually involve positioning the cursor and then issuing the command.

Nearly all terminals have a clear screen function, and we require it.

We expect carriage return and line feed to work on the display just as they work on a standard teletypewriter device. That is, carriage return moves the cursor to the left margin and line feed moves it to the next line without changing the character position on the line. Although a single "next line" code is often useful, we do not feel it is reasonable for a manufacturer to omit either of these two fundamental functions.

When a user selects a character on the screen by pointing to it with the mouse tracking spot and pushing a button, the applications program usually responds by "marking" that character for confirmation. The "marking" feature is very important to the interaction process and it has given us a bit of trouble in transforming an alphanumeric display into an effective workstation. We will describe the problem in some detail.

The "marking" is done by altering the appearance of the character without obliterating it. If the user selected the wrong character he will abort (or "back out of") that selection and select another. The NLS program responds by removing the first mark and putting up another. Hence, it must be possible to "mark" and "unmark" characters on the screen without altering the text in any other way and without rewriting the character. On displays where single characters can be made to blink, reverse in video, or change in some other way, marking will be implemented using that feature.

But very few displays have the capability of "marking" a character without using up a character position on each side of the character, which is unsatisfactory for our purposes. We have had success by showing the "mark" by "flashing" the cursor at the marked character position at a rate of about three times a second; the cursor is returned to the tracking spot between flashes. This is satisfactory if the display-Line Processor connection is of high enough baud rate (4800 or 9600 baud) and the cursor positioning within the display is fast and does not result in extraneous flashes on the screen.

Clearly, the former type of "marking" is preferred because the latter scheme results in a flashing tracking spot and becomes confusing when there are several "marks" on the screen. Probably the most desirable type of "marking" feature would be to make the character blink at about 3 cps.

Occasionally, it is desirable to write a string on the display with a special appearance, to get the user's attention. We call this "standout mode." Most displays have some kind of standout feature such as blinking, underline, high intensity or reverse video—we require some kind of standout mode. This differs from marking: in marking an existing character on the screen is altered without rewriting, and in standout mode new text is written on the screen.

Of course, all these functions take some finite time for the display to carry out. We would like to see all functions performed as fast as possible. In particular, we expect that all functions except delete line and clear screen will be performed in one millisecond or less. We expect delete line and clear screen to be done in about 5 to 7 milliseconds. We set an upper bound of 120 milliseconds for any function execution time.

2. User Considerations

We consider 24 lines by 64 characters a minimum adequate screen size; however, 27 lines by 80 characters is much more useful. The most desirable would be a full text page of 66 lines by 80 characters. No matter what the screen size, we expect the terminal to have enough memory capacity to nearly fill the screen with text.

The terminal must display the full ASCII character set, including upper and lower case letters.

The keyboard should be a standard typewriter style keyboard with the full ASCII character set, including control characters. Some form of key rollover feature and a comfortable feel are very important.

The display output should be readable, easy to look at and flicker free. Our applications are geared toward comfortable day-long use in an office. The terminal should be absolutely quiet and a pleasant thing to work with in all respects.

HARDWARE DESCRIPTION

The organization of the Line Processor is outlined in Figure 5. The Line Processor is discussed from a hardware standpoint in Reference 8.

The heart of the device is an Intel 4004 CPU, which is a four-bit parallel microcomputer in a single MOS integrated circuit chip. The program resides in up to six Intel 1702A programmable ROM (PROM) chips. Each PROM contains 256 8-bit bytes. These PROM chips are mounted in sockets and can be removed, erased and rewritten in a few minutes.

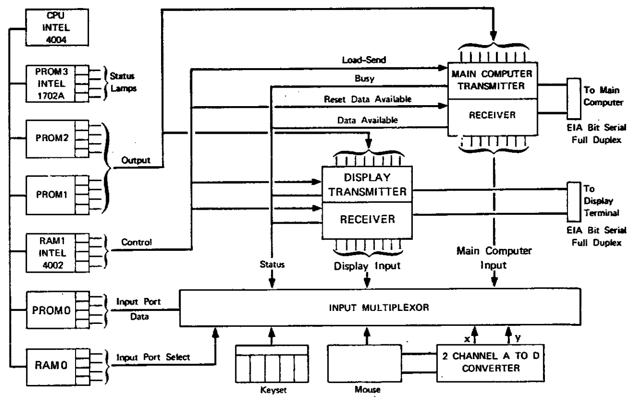


Figure 5—Microcomputer and I/O configuration

The read/write memory consists of 320 four-bit bytes packaged in four Intel 4002 RAM chips. These Intel chips are part of Intel's MCS-4 Micro Computer Set, and are easily connected. Both the RAM and PROM chips have I/O ports that are addressed like memory, but they are accessed with different CPU instructions.

The input devices are multiplexed into one input port. A four-bit address on another port selects the particular four bits of input to be read. The output devices (two serial transmitters) are loaded via two ports and controlled via another port.

The general configuration is such that Intel 4001 ROM chips can be substituted for the PROM chips, if the production quantity warrants the initial cost of cutting the ROM masks.

The total cost of all parts is about $1200 per unit in single quantities. We intend to find a firm that will manufacture Line Processors and provide nation-wide maintenance. A firm is currently making them for us on a limited basis for $1800 each.

CONCLUSIONS

The Line Processor approach can effectively upgrade a peripheral device and amplify its capabilities with no actual modifications to the device itself. At the same time, it can optimize the interface between the device and the main computer complex, with respect to both hardware and software.

In our application of NLS workstation development, the concept has been beneficial. The alphanumeric terminal needs no modification, and the main computer does not need to know the true nature of the workstation. In other applications, new peripherals could conceivably be interfaced to existing software without modifications to either.

A Line Processor coupled with a satisfactory alphanumeric video display terminal results in an NLS workstation that is as effective as a high speed general CRT workstation, except for one respect: the inability to arbitrarily move text on the screen without rewriting it. But, intelligent display manipulation algorithms reduce this problem to the point that users hardly realize that it exists.[2]

Other applications for the Line Processor concept are appearing. A microcomputer-based device is being developed at the University of California, Santa Barbara, to make a PLATO IV terminal appear to be a general purpose ASCII terminal for ARPA Network Use. At ARC, we are considering using a simplified Line Processor to make a line printer with a difficult interface appear to be a serial ASCII device, to avoid purchasing an expensive controller.

A prototype Line Processor coupled to a Delta Data 5200 display terminal has been in operation at ARC and over the ARPA Network since September 1973. The microprogram is 1024 8-bit bytes long and operates with the main computer connection set from 300 to 2400 baud. More recent versions operate at 4800 baud with either a Delta 5200 terminal or a Hazeltine H2000 terminal.

We are currently expanding the microprogram to provide more services to the user. For example, we are providing a hard copy output connection to the Line Processor to allow the user to obtain a printout at the same time he uses the workstation for unrelated matters.

We expect the microcomputer equipment to become cheaper and faster. These developments will allow the Line Processor and similar devices to have more capabilities. It appears that such devices, if properly designed, could tend to reduce software problems and expenses.

A growing trend, brought about by computer networks and interconnections of various kinds, is to divide workloads and define appropriate interfaces. With this method, the computing takes place over several processors, hopefully each best suited to its workload. We have accomplished that with the Line Processor. We are in the infancy of a "distributed computing" era in which microcomputer devices such as the Line Processor will clearly have a growing role.

ACKNOWLEDGMENTS

The development of the Line Processor has been a team effort by several members of ARC. The hardware design and prototype construction were done by Martin Hardy and Rodney Bondurant. Charles Irby and Kenneth Victor programmed NLS to use Line Processor workstations and modified our TENEX for the Input protocol. The design of the display manipulation protocol and the micro-programming were done by the author. Dr. Richard Watson oversaw the effort and was a primary source of useful criticisms during the writing of this paper. We all owe many thanks to Delta Data Systems for extended use of a Delta 5200 video display demonstrator during development of the Line Processor.

The work reported here is currently supported primarily by the Advanced Research Projects Agency of the Department of Defense, and also by the Rome Air Development Center of the Air Force and by the Office of Naval Research.

REFERENCES

1. Engelbart, D. C., R. W. Watson and J. C. Norton, "The Augmented Knowledge Workshop," *AFIPS Proceedings*, National Computer Conference, June 1973, (SRI Catalog Item 14724).
2. Irby, C. H., "Display Techniques for Interactive Text Manipulation," *Proceedings of the National Computer Conference*, May 1974, (SRI-ARC Catalog Item 20183).
3. Roberts, L. G. and B. D. Wessler, *The ARPA Network*, Advanced Research Projects Agency, Information Processing Techniques, Washington D. C., May 1971, (SRI-ARC Catalog Item 7750).
4. Bobrow, D. G., et al., "TENEX, A Paged Time Sharing System for the PDP-10," presented at *ACM Symposium on Operating Systems Principles*, 18-20 October, 1971, Bolt Beranek and Newman Inc., 15 August 1971, (SRI-ARC Catalog Item 7736).
5. English, W. K., D. C. Engelbart and M. A. Berman, "Display-Selection Techniques for Text Manipulation," *IEEE Transactions on Human Factors in Electronics*, Vol. HFE-8, Number 1, pp. 5-15 March 1967, (SRI-ARC Catalog Item 9694).
6. Engelbart, D. C., "Design Considerations for Knowledge Workshop Terminals," *AFIPS Proceedings*, National Computer Conference, June 1973, (SRI-ARC Catalog Item 14851).
7. Engelbart, D. C. and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings*, Vol. 33, 1968, (SRI-ARC Catalog Item 3954)
8. Hardy, M. E., "Micro Processor Technology in the Design of Terminal Systems," *Proceedings of the IEEE COMPCON*, 1974, (SRI-ARC Catalog Item 20185).

APPENDIX: LINE PROCESSOR PROTOCOL AND OPERATION

The Display Manipulation protocol calls for the transmission of unescorted characters and short command strings. Command strings begin with an escape character: we use 34 octal. Characters within the command strings are seven bit ASCII printable characters. Sending printable characters, where possible, rather than control characters makes debugging and troubleshooting much less painful.

The Line Processor operates in one of two modes. One mode simulates a teletypewriter. The other is the normal display mode that allows NLS display manipulation.

The mode is specified by mode set commands from the main computer. The Line Processor responds to the "enter display mode" command (also called the interrogate command) by sending a string in protocol format that informs the main computer of the display screen size, length of time it takes to delete a line and the baud rate; the last two parameters are important for timing considerations.

To make the displays function correctly, it is necessary to send the proper number of "padding" or null characters while the display performs involved functions, like delete line and clear screen.

The microcomputer is programmed to send the padding characters, but it has limited buffer space and must receive pads from the main computer as well. It would be fine to have the main computer refrain from sending anything in most applications, but our users will frequently be connected by way of a computer network. In such an environment, the only way to ensure the proper timing is to send the appropriate number of padding characters.

The number of padding characters that need to be sent depends on the length of time it takes the display to perform the function and the baud rate going into the Line Processor. So, we have installed a baud rate switch on the Line Processor that can be read by the microcomputer.

Hence, when responding to the "enter display mode" command, the microcomputer includes the baud rate setting along with the other terminal parameters. From the baud rate and the delete time parameter, the applications program can compute the number of pads to be sent.

Display Manipulation Protocol Primitives

POSITION CURSOR (X, Y)
This command positions the cursor to the designated spot and stops the mouse tracking process. Any subsequent unescorted characters are written on the display starting at the cursor position.

RESUME TRACKING
This command is used after positioning the cursor and writing a string, to start the mouse tracking process again. Note: A string is written by sending: POSITION CURSOR; (the string); RESUME TRACKING.

DELETE LINE
The line at which the cursor has been positioned, is deleted.

INSERT LINE
A new line is inserted after the line on which the cursor was positioned.

CLEAR SCREEN
The entire screen is cleared.

BEGIN STANDOUT MODE
All text written on the screen subsequent to this command will be in "standout mode."

END STANDOUT MODE
This returns the text writing mode to normal.

RESET
This command resets the line processor to normal mode and clears the screen.

WRITE A STRING OF BLANKS (N = number of blanks)
It is useful to be able to clear a short area of the screen with this command.

PUSH BUG SELECTION (X, Y)
The coordinates for the bug selection mark are pushed on a stack, and the indicated character is marked.

POP BUG SELECTION
The top entry on the bug selection stack is removed and the mark at the corresponding screen position is removed.

SPECIFY TTY-SIMULATION WINDOW (Y1, Y2)
Y1 and Y2 specify the top and bottom line for the scrolling window. Any subsequent TTY-type output will be scrolled in this window.

ENTER DISPLAY MODE (INTERROGATE)

ENTER NORMAL MODE

The GRAFIX I image processing system

by ARNOLD K. GRIFFITH

Information International Inc.
Los Angeles, California

INTRODUCTION

The GRAFIX I system was developed in the late 1960's as a fast flexible system for processing and analyzing filmed images, particularly of material which is essentially binary (black and white), such as printed text, line drawings, certain biomedical images, fingerprints, etc. It incorporates a large scale general purpose time shared computer to provide the facilities for the efficient development of algorithms necessary to perform various image processing and analysis tasks. In addition it contains a fast, high resolution flying-spot film scanner and a powerful and rather general slave processor (the binary image processor, or BIP) which provide data collection and manipulation facilities adequate to perform image processing and analysis tasks at commercially practical speeds. So far it has been successfully employed in a commercial environment to the reading of printed multifont text in complex page formats, and to the reading of Cyrillic, Greek and even handprinted text. At present we are considering future applications beyond the area of optical character recognition, particularly the analysis of engineering drawings, as well as the automatic analysis and classification of fingerprints, the analysis of biomedical images such as chromosomes, the analysis of x-ray images, and the analysis of satellite imagery data, among others.

THE GRAFIX I SYSTEM

The main processor in the GRAFIX I is a large, third generation computer with 144,000 words of 36-bit two microsecond memory. It operates under a standard time sharing monitor and incorporates the usual range of text editors, file handlers, assemblers and compilers. Most of the developmental programming is carried out in TRIP, our own interactive on-line compiler language. In addition to the standard peripherals such as magnetic tape drives, "Microtape" drives, disk drive and a high speed line printer, the hardware includes a sonic data tablet for the input of graphical information, a number of teletypes, and five terminals with full keyboards and CRT's which display both images and characters.

The flying spot scanner reads rectangular rasters of density values from filmed images at a rate of up to 500 points per millisecond. The number and spacing of points in both axes, the position and orientation of the scan raster, the dwell time of each point, and the spot size are all under program control. Points on an image are specified in a 15-bit per-axis coordinate system which allows a very precise angular and incremental control of the position of the raster scan relative to the image. The scanner has extremely high positional linearity and repeatability, and resolves about 6000 points across the CRT face. Density values are measured on a range of 0 to 2.5 (100 percent to 0.5 percent transmission) on a logarithmic scale of 512 values.

The slave processor, the Binary Image Processor (BIP), performs a wide variety of processes and measurements on images at core memory limited speeds. It will be described in detail in the next section.

THE BINARY IMAGE PROCESSOR

The GRAFIX I system is designed principally to operate on "binary images," that is, images which are essentially black or white, such as engineering drawings, printed pages, fingerprints or waveform photographs; or images such as bubble, chamber, cell, or chromosome photographs, which for many purposes may be reduced to black and white images without significant loss of information. These images are represented in the GRAFIX I system as arrays of zero- and one-bits in the 36 bit words of the system's main memory. They are manipulated by a special high speed slave processor, the binary image processor (BIP), which has direct access to any part of the main computer memory.

Before discussing the capabilities of the BIP in detail, I shall present a simple paradigm of what is meant by binary image processing in the present sense: Consider a set of n sequential 36-bit words of core memory and let the bits of the ith word be indexed by j from 1 to 36, so that the value of the jth bit of the ith word is represented by $a_{i,j}$. These values form a matrix $| a_{i,j} | \; i=1 \ldots n; j=1 \ldots 36$ of ones and zeroes. Consider two such matrices $| a_{i,j} |$ and $| b_{i,j} |$, and consider the matrix $| c_{i,j} |$ where:

$$c_{i,j}=F(a_{i,j}, b_{i,j}, b_{i+1,j}, b_{i+1,j+1}, b_{i+1,j-1}, b_{i,j+1}, b_{i,j-1}, b_{i-1,j}, b_{i-1,j-1}, b_{i-1,j+1})$$

for some function F. Clearly this forms a binary image with the same dimensions as those represented by $| a_{i,j} |$ and $| b_{i,j} |$, provided of course that some ad hoc value is assigned to such values as $b_{-1,-1}$, etc. Considered as a transformation or combination of images, the production of this third image from the first two is local in the sense that the value of some point in the result image is dependent only on the corresponding point in the first image and the "local neighborhood" (of radius 1) of the point in the second image. Now if this transformation were purely local in the sense of a point in the result image being a function only of the values of the image at the corresponding points (i.e. if $c_{i,j}=F(a_{i,j}, b_{i,j})$), then it would be quite uninteresting, since actually there would be only sixteen possible functions corresponding to "or-ing," "and-ing," negating the first image irrespective of the second, etc. However, the inclusion of neighborhood bits of the second image as arguments to a possible image processing function allows a total of 2^{1024} possible functions!

The BIP is in most respects more general than the paradigm just described. It has addressing features which allow either image to occupy only a predetermined subset of contiguous bits within an image; and it is relatively easily programmed to allow the image height to be greater than 36 bits. The major respect in which the BIP is not as general as the paradigm discussed is in that there are only 33 bits in the control field specifying the function, and hence only 2^{33} possible functions. The set of functions is by no means an arbitrary subset of all possible functions. It was in fact chosen on the basis of a rather extensive theory and practical considerations worked out by Gray[1-3] and others, for the purpose of providing a rich variety of functions, as will be seen later in this section.

A second major area of BIP function not mentioned by the paradigm is that of global measurements. In the process of generating a result image the BIP can compute the area, i.e., number of ones; together with the number of configurations of subsets of the result image of the form $c_{i,j}$, $c_{i+1,j}$, $c_{i,j+1}$, $c_{i+1,j+1}$ (i.e., subsquares of the result image) in which all four bits are one, in which three bits are one, in which non-diagonal pairs of bits are one, and in which only one bit is a one. From these values and others, again according to the theory worked out by Gray, the Euler number (number of objects minus the number of holes), the approximate line width, as well as a number of other global features, such as average slant of lines, may be calculated. A final global measurement of this sort is the area of the "exclusive-or" of two images, i.e., the area in which they are not alike, which is a measure of similarity. A single pass over a pair of images not only computes this area of dissimilarity but also simultaneously computes eight similar values corresponding to one of the images being displaced relative to the other by one unit in each of eight possible directions.

An additional feature of the BIP is a limited capability to work with arrays of integral values, not just one or zero. The two images to be processed consist of six bit bytes, packed six per word, and would produce a binary image with ones where one of the corresponding pair of values exceeds the other, and zeroes elsewhere. If one image is an array of constant values, then this process amounts to thresholding the other image at that value. Having one image of smoothly varying value allows position dependent thresholding.

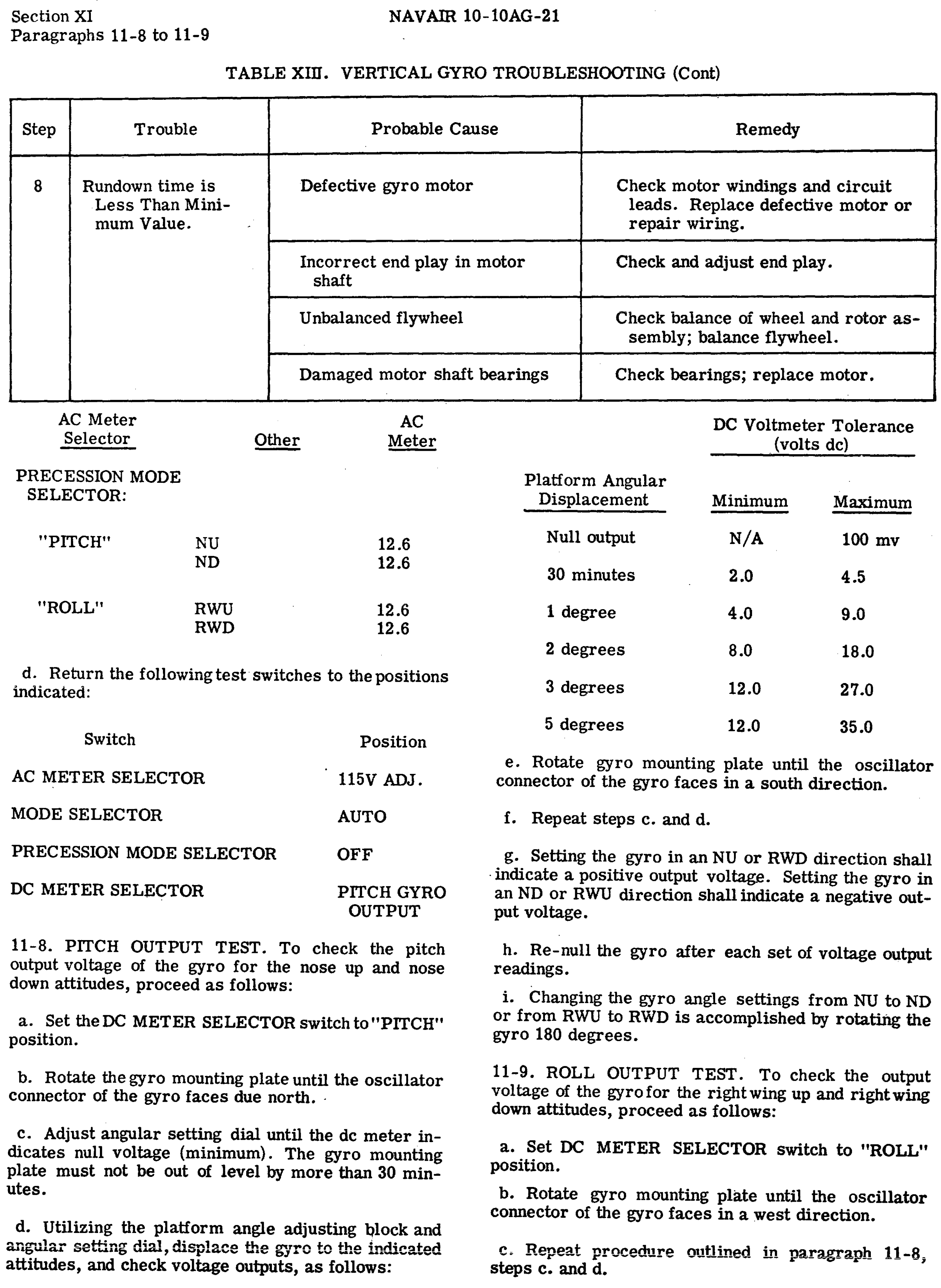
The purpose of the BIP is to serve as a special purpose slave processor to perform inner loop tasks in image processing operations at high speeds (40 MHz), leaving system control, decision making tasks, etc., to the general purpose computer operating the system. The BIP operates at speeds of up to 1000 times that possible on a standard high speed computer. Its use of modern integrated circuits and pipeline construction allows it to run at speeds of 25 nanoseconds per image point. A process operating on an array image of 36 bits high runs at memory limited speeds even in a one microsecond memory. The speed of complex processes is proportional to the area of the image and to the number of passes required to perform it.

MAN COMPUTER SYMBIOSIS IN THE READING OF COMPLEX PRINTED PAGES

Recently the GRAFIX I has been employed to perform a number of major commercial text conversion (OCR) tasks. It has been our experience that "real-world" text conversion does not amount simply to the character by character or word by word recognition of printed text. Rather it has included the necessity of recognizing certain typographical features of the pages such as the location of text relative to illustrations or the structure of text within tables. In addition such tasks often require that the recognized text be broken into fields according to content and sometimes suitably formatted in instances when the converted text is subsequently manipulated by an information retrieval system, or output and reformatted by a computer typesetting device. Generally it is beyond the state of the art in artificial intelligence to perform all of these tasks automatically.

We have structured our text conversion systems to perform the various format recognition, character recognition and output formatting tasks as automatically as possible. These text processing systems, however, are programmed to be "aware" of their own limitations, and to ask for human help when unable to cope with the complexities of the data with which they are presented. We have found this technique of "man computer symbiosis" to be an extremely powerful one, providing as it does an efficient division of labor between man and computer.

Conversion by GRAFIX I of technical manuals affords a good illustration of a text conversion process in which typographical features of the text must be taken into account. Figure 1 illustrates a typical page which the system converted, in its entirety, in a recent procurement benchmark test. Information critical to successful reading of the page included the location of the heading and page numbers at the top and bottom left, the locations of the various rules of the table at the top of the page, the locations of the tab stops in the three passages of tabulated material in the running text columns, and the locations of the portions of



Section XI
Paragraphs 11-8 to 11-9

NAVAIR 10-10AG-21

TABLE XIII. VERTICAL GYRO TROUBLESHOOTING (Cont)

Step	Trouble	Probable Cause	Remedy
8	Rundown time is Less Than Minimum Value.	Defective gyro motor	Check motor windings and circuit leads. Replace defective motor or repair wiring.
		Incorrect end play in motor shaft	Check and adjust end play.
		Unbalanced flywheel	Check balance of wheel and rotor assembly; balance flywheel.
		Damaged motor shaft bearings	Check bearings; replace motor.

AC Meter Selector	Other	AC Meter
PRECESSION MODE SELECTOR:		
"PITCH"	NU	12.6
	ND	12.6
"ROLL"	RWU	12.6
	RWD	12.6

d. Return the following test switches to the positions indicated:

Switch	Position
AC METER SELECTOR	115V ADJ.
MODE SELECTOR	AUTO
PRECESSION MODE SELECTOR	OFF
DC METER SELECTOR	PITCH GYRO OUTPUT

11-8. PITCH OUTPUT TEST. To check the pitch output voltage of the gyro for the nose up and nose down attitudes, proceed as follows:

a. Set the DC METER SELECTOR switch to "PITCH" position.

b. Rotate the gyro mounting plate until the oscillator connector of the gyro faces due north.

c. Adjust angular setting dial until the dc meter indicates null voltage (minimum). The gyro mounting plate must not be out of level by more than 30 minutes.

d. Utilizing the platform angle adjusting block and angular setting dial, displace the gyro to the indicated attitudes, and check voltage outputs, as follows:

Platform Angular Displacement	DC Voltmeter Tolerance (volts dc) Minimum	Maximum
Null output	N/A	100 mv
30 minutes	2.0	4.5
1 degree	4.0	9.0
2 degrees	8.0	18.0
3 degrees	12.0	27.0
5 degrees	12.0	35.0

e. Rotate gyro mounting plate until the oscillator connector of the gyro faces in a south direction.

f. Repeat steps c. and d.

g. Setting the gyro in an NU or RWD direction shall indicate a positive output voltage. Setting the gyro in an ND or RWU direction shall indicate a negative output voltage.

h. Re-null the gyro after each set of voltage output readings.

i. Changing the gyro angle settings from NU to ND or from RWU to RWD is accomplished by rotating the gyro 180 degrees.

11-9. ROLL OUTPUT TEST. To check the output voltage of the gyro for the right wing up and right wing down attitudes, proceed as follows:

a. Set DC METER SELECTOR switch to "ROLL" position.

b. Rotate gyro mounting plate until the oscillator connector of the gyro faces in a west direction.

c. Repeat procedure outlined in paragraph 11-8, steps c. and d.

Figure 1

the running text columns consisting of "pure" text. The information was entered by a human operator using a data tablet, together with a program specifically designed to enter such data.

A second example of human-computer symbiosis is the "reject processing" procedure employed in all our optical character recognition applications. The character recognition system is so programmed that it either decides upon the identity of an individual character, or decides that it is unable to recognize it. In the latter case, the image of the character is stored within the output stream of recognized text. Immediately after a batch of pages has been processed by the recognition procedure it may be run through a second off-line program in which each unidentified ("rejected") image is displayed on a CRT screen, together with the line of recognized text characters containing it, for a human operator to identify by typing the appropriate character at the keyboard. The human recognition is performed both on the basis of the appearance of the character image and on the basis of the textual context. Typically such reject processing can be performed by an operator with a few days' training at a rate of 3000 characters per hour.

An example of man-computer symbiosis involving the formatting and fielding of text after reading is provided by our conversion of a large number of U.S. patents into a form suitable for interrogation by an information retrieval system. Patents consist of up to 35 different types of items, including patent number, date, assignee, summary, claims, etc. Each had to be separated and identified and in many instances specially formatted. This was accomplished by an interactive heuristic program operated by a human seated at a display console. Each item was automatically searched for on the basis of keywords and phrases which were not totally reliable due to the wide variations in structure and phrasing of the component parts, and the presence or absence of particular parts within an individual patent. After an attempt was made to find a particular item, the item, together with a large portion of the surrounding text was displayed on the screen with pointers around the item in question. At this point the operator could simply indicate approval by typing a control character, or could reposition the pointers in case the finding heuristics had failed.

The exploitation of the technique of man-computer symbiosis is a direct consequence of our use of a full sized computer with time sharing and extensive system software. The development of the last application, for example, took only three man months of programming. Besides facilitating the development of these procedures, the full scale system was necessary for their post-development implementation.

HEURISTICS

It should not be inferred from the previous section that the only way "intelligent" behavior is manifested by the various GRAFIX I OCR programs is by receiving the appropriate instructions from a human operator. On the contrary, many of the various OCR sub-processes operate autonomously, performing quite intricate analyses with no provision whatever for human interaction and prompting. These procedures are carried out by elaborate heuristic programs developed by the staff which are constantly undergoing further development and refinement. The use of extensive heuristic software was a conscious part of the original design of GRAFIX I; and the rapid design and development of such programs is made possible by the use of a time shared computer with extensive software support.

An example of our use of heuristic procedures is the handprinted character recognizer currently under development by the present author. Most handprinted character recognition procedures currently described in the literature[4-7] tend to be more "algorithmic" than heuristic: Typically a set of numerical "features" or some sort of formal description is derived from a character image by an analysis which performs a large amount of calculation independently of the nature of the character image. This feature vector, or formal description, is then given to a decision algorithm which decides on the identity of the character (or possibly rejects it as unrecognizable) by means of a linear decision procedure or a feature vector table lookup process, or a formal linguistic analysis in the case of a formal description. Such procedures often have the disadvantage, when applied to data as variable as handprinted characters, to occasionally yield wildly unpredictable results. In addition, attempts to modify such algorithms to perform better in certain respects often lead to a decrease in performance in other respects. (This, and other arguments are particularly applicable to the linear decision or "perceptron" technique, see Minsky and Papert.[8]) Although a number of systems organized along these lines have produced respectable results, the present system, in an attempt to circumvent some of these difficulties, has been designed and organized along purely heuristic lines. In particular it is composed of a large number of simple, carefully designed, and well-understood sub-tests. These are arranged into a hierarchy in such a fashion that tests at various levels are usually only performed when relevant or "necessary," and so that the whole structure may be quite simply and selectively modified to improve in a particular respect without impairing its performance in other respects. On one sample of about 13,000 mixed alphanumeric characters (26 letters and 10 numerals), printed by untrained clerks, it achieved a reject rate (unable to recognize) of around 4 percent, and a substitution rate (misidentification) of around 0.3 percent.[9]

Numerous other examples of heuristic procedures are embodied in the GRAFIX I machine printed text reading system. Two examples discussed in the previous section, although involving human interaction, include quite elaborate heuristics to perform those portions of their respective tasks that are performed independently of the human operator. The procedure employed in the technical manual conversion process to read text in tabular form, after receiving information from a human operator as to the location of the rules in the table, must re-find these rules more exactly on the filmed image, and employs a rather elaborate line and character finding procedure to properly read the material within each ruled box. In the case of the fielding and formatting of patents, a number of heuristics are used to tentatively locate each item in the converted patent text before the human operator is consulted. In most cases the heuristics produced a correct result; and the operator's only task was to indicate correctness. A number of other procedures from the basic OCR system, such as "page finding," "line finding" and character finding similarly rely heavily on heuristics. Some of these procedures will be discussed in the next section in connection with the concept of heterarchical organization.

HETERARCHY AND HIERARCHY

The OCR system currently implemented on the GRAFIX I system is principally hierarchical in structure. The first step in processing a frame of text consists of finding the outline of the page image on the film. The text on a page is considered to be composed of a number of "fields" consisting of one or a number of lines of text, a title, an entry in a piece of tabulated material, etc. Information about the fields of a particular page is supplied from a data file of "descriptors" entered either manually or with a data tablet. Each field is read line by line and each line is read character by character. Each line is first found, then read; and within a line each character is first found and then read.

The procedure just described is a prototypical hierarchy, where each step of the processing depends on the accuracy of decisions of the previous step. For example once a field is found, it is assumed that the finding procedure is correct, and the success or the failure of the finding and reading of the individual lines is dependent on the accuracy of the field finding data. No attempt is currently made to re-find the field if attempts to find the constituent lines within it meet with failure.

Recent research in the analysis of complex scenes (e.g., [10,11]) has explored a more general approach than strictly hierarchical organization in dealing with real world or complex images. Clearly it is an essential limitation for information pass from level to level in one direction only, as for example in the case of scene analysis, from local edge detection to edge line and contour detection, to detection of simple forms, to detection and recognition of complex objects. In the case of optical character recognition of text on elaborately structured pages, this limitation might take the form, as previously mentioned, of making a "one-shot" decision as to the location of a field of text and then forcing the line and character finder to accept this without complaint, without letting a failure in line finding pass back to the field finder, if necessary, to force a re-find of the field. Some of these limitations are circumvented by a more general approach to program organization, termed "heterarchical," which allows a two way interaction between levels to occur. In effect the program interacts with the data at hand instead of just analyzing it step by step to higher levels of abstraction or in greater levels of detail.

Heterarchical program organization has been employed to advantage in a number of sub-processes of the present GRAFIX I OCR system:

(1) The image of a line of text is scanned from left to right in a series of "segments" of about seven characters in length. Given the proper location of the first segment of a line, the approximate locations of subsequent segments for that line may be inferred using information as to the orientation of the page and the orientation of other lines of text on the page. However lines of text are sometimes not exactly straight nor are they always oriented exactly parallel to each other. Furthermore, in the case of the first line on a page, the only information is the orientation of the borders of the page, which is a poor predictor of the orientation of the lines of text on it. Consequently it is generally impossible to scan a line of text segment by segment using only global orientation information and the correct location of the left end without risking cutting off the tops or bottoms of characters within certain segments. Our approach to this problem is as follows: In the course of recognizing the characters from a particular segment, the positions of the baselines of these characters relative to the bottom of the segment are computed for each character recognized. The baseline is a function not of, e.g., the bottom of the character image, but is computed after recognition on the basis of a fiducial line within the character (often at the bottom, but not always) whose location is dependent on the identity of the character. The difference between some predetermined constant and this baseline-to-segment-bottom distance is then fed back to the scanner and used to define the exact vertical location of the next scan segment.

(2) The process of reducing a gray level image as obtained from the scanner into a binary image to present to the recognition procedure is accomplished by thresholding the gray level image at some clip level. Because of blurring and other effects, the appearance of the resulting binary image, and hence the recognizability of the characters it contains, is affected by this clip level. Due to variabilities of the image resulting from such factors as variations in the reflectance of the ink and paper of the original document, it is in general impossible to choose a single threshold which will be appropriate for an entire page image. Our approach has been to initially threshold a new image at the same threshold which was successful for the preceding segment. In the course of subsequent analysis, the apparent linewidth of the characters in the segment is calculated. If the width is outside certain tolerances, no further recognition analysis is performed on the binary image of the segment. Instead the information regarding the degree to which linewidth is outside

tolerances is fed back to the thresholding procedure, and the gray level image of the segment is re-thresholded accordingly. This procedure may be repeated several times.

(3) In certain instances a line of text may be entirely unrecognizable due to being in an unexpected font, or being systematically degraded along its entire length. Such situations are generally impossible to predict a priori; but when they are encountered it would be a waste of effort to apply the recognition process to every character in the line. This is especially the case because with unrecognizable material, the recognition procedure has "tried everything" before it gives up and may expend ten times as much effort on a rejected character as on a recognized one. To overcome this problem the GRAFIX I OCR system has a provision whereby if a certain number of rejects is encountered in a line, the line is re-read using a different set of character masks, or is abandoned if no other sets of masks remain to be tried. To see how this procedure may be considered to be heterarchically organized, consider the task of the recognition of all the characters in a particular line of text. The restart of this task with a new "font," or the abandonment of the task if all fonts have been tried, is essentially a higher level decision based on information (the failure of the lower level process of recognition) being passed up to it, which in turn affects other lower level processes (the further recognition of characters within the line).

FUTURE DIRECTIONS

The GRAFIX I was conceived as an image processing system of considerable versatility. Its application to date principally as an optical character recognition (OCR) device has been prompted by the commercial promise of this area. Having realized a substantial portion of the potential of the GRAFIX I system in OCR, we are considering further applications, particularly in the areas of the processing of essentially binary images. An area in which we hope to become involved in the near future is the intelligent reading of engineering drawings and technical diagrams. Recent work of the present author[10-12] and others at MIT[13,14] as well as at Stanford,[15] SRI and the University of Edinburgh in the analysis of scenes consisting of prismatic solids, appear to be adaptable to the problem of efficiently extracting the component lines from these drawings so as to represent them in a reasonably compact and updatable form. Other possible areas include: biomedical image processing such as cell counting, chromosome analysis, etc.; high volume metallurgical image processing such as fibre measurement, inclusion counts, grain size, etc.; automatic inspection of x-ray photographs and photographs of manufactured parts for defect analysis; the analysis of fingerprints; and others.

ACKNOWLEDGMENTS

The author is particularly indebted to Stephen Gray for aid in the preparation of this paper. In addition the following, among others, have significantly contributed to the design and implementation of the GRAFIX I: Russell Ham, Dick Martin, Dan Forsyth, and Ed Fredkin.

REFERENCES

1. Gray, S. B., *Technical Description of The GRAFIX I Image Processing System*, Document 90340, Information International Inc., Los Angeles, Calif. 1971.
2. Gray, S. B., "Local Properties of Binary Images in Two Dimensions," *IEEE Transactions on Computers*, May 1971.
3. Gray, S. B., *The Binary Image Processor and its Applications*, Document 90365, Information International Inc., Los Angeles, Calif., January 1972.
4. Rholand, W. S., P. J. Traglia, and P. J. Hurley, "The Design of an OCR System for Reading Handwritten Numerals," *Proceedings of the Fall Joint Computer Conference*, 1968.
5. Casky, D. L., and C. L. Coates, *Machine Recognition of Handprinted Characters*, Technical Report 126, Information System Research Laboratory, Electronics Research Center, The University of Texas, Austin, May 1972.
6. Munson, J. H., "Experiments in the Recognition of Hand Printed Text: Part I—Character Recognition," *Proceedings of the Fall Joint Computer Conference*, 1968.
7. Katsuragi, S., H. Genchi, K. Mori, and S. Watanabe, "Recognition of Handwritten Numerals Using Decision Graph," *Proceedings of the First International Joint Conference on Artificial Intelligence*, May 1969.
8. Minsky, M. L. and S. Papert, *Perceptrons*, MIT Press 1968.
9. Griffith, A. K., *Preliminary Results of the Handprint Recognition Verifier's Performance on Recently Acquired Data*, Information International Technical Memo 197, November 1973.
10. Griffith, A. K., "Edge detection in Simple Scenes using a Priori Information," *IEEE Computer Transactions*, April 1973.
11. Griffith, A. K., *Computer Recognition of Prismatic Solids*, M.I.T. Project MAC Technical Report MAC-TR-73, August 1970.
12. Griffith, A. K., "Mathematical Models for Automatic Line Detection," *JACM* January 1973.
13. Guzman, A., "Decomposition of a Visual Scene Into Three-Dimensional Bodies," *Proceedings of the Fall Joint Computer Conference*, 1968.
14. Minsky, M. and S. Papert, *Progress Report 1968-1969*, M.I.T. Project MAC Artificial Intelligence Memo 200, M.I.T., Cambridge-Mass., 1970.
15. Feldman, J. A., et al., "The Stanford Hand-Eye Project," *Proceedings of the First International Joint Conference on Artificial Intelligence*, May 1969.

Hardware/software design considerations for high speed/low cost interactive graphic communication systems

by THOMAS L. BOARDMAN, JR.

University of Michigan
Ann Arbor, Michigan

A BIT OF HISTORY

Graphics has been among the most exciting areas of the computer business since its dramatic appearance in the early 1960's (Sketchpad[1]). There have been many significant applications throughout NASA and the aerospace industries, some large companies, and even some sponsored university projects. These have, however, utilized high cost graphic equipment, connected directly to large scale (CDC 6000 or IBM 360/50+) mainframes often on a dedicated basis. The terminal and mainframe costs precluded general use in small companies or universities.

In 1969, several terminal manufacturers made available CRT displays suitable for text and vector graphics for under $10,000. This marked the beginning of "poor man's" graphics. But these displays were originally designed and utilized as "fancy" teletypes. They were connected to mainframes at teletype (TTY) speeds and forced to use escape character switches allowing the TTY characters to represent both text and vector information. Mainframe hardware and software were designed around the inherently low speed, character (or at best line) at a time operation of teletype terminals. The system changes necessary to adequately support high speed, image-oriented terminals were understandably resisted.

The timesharing system developers and suppliers justified this resistance, arguing that use of low cost graphic terminals would not really be effective until high speed communication (10 to 100 times teletype speeds) was possible. This, they further contended, was not feasible due to the high modem and phone communication costs. They therefore refused to support graphic terminals on any useful scale. Now, however, AT&T's recent decisions[2] to offer more complete communications services along with reduced modem costs brought on by the general circuit component trends, are making higher speed communication economically feasible. The time has come to provide timesharing systems which can support this high speed, graphics-oriented communication.

INTRODUCTION

The design considerations involved in connecting many low cost terminals to a single mainframe at speeds suitable for interactive graphics are discussed in this paper.* There are several overriding design constraints.** First, the dollar investment in interface and multiplexing hardware should not exceed ten percent of the terminal costs. Second, the full range of graphics devices (incremental and electrostatic plotters, storage tube terminals, refresh terminals, sophisticated image processing systems, . . .) must be supported allowing high volume batch through highly interactive graphic use of the mainframe. Finally, and perhaps most important, the communication system must interface to existing operating systems with a minimum of mainframe software modification and minimum central processor (CPU) overhead. These constraints require a data concentrator minicomputer between the terminals and the mainframe.

The major components of the data concentrator system are indicated in Figure 1. Component designs are detailed in the sections which follow. It must be emphasized that this paper deals with a communication system, NOT a timesharing system. It discusses a means of transferring information at sufficient speed, with low enough cost and CPU overhead, to allow graphic terminals to communicate effectively with an existing timesharing system.

HOW FAST?

The first sections of this paper contend that interactive graphics is not practical at teletype speeds (110 to 300 baud). A reasonable question follows: how fast is fast enough? The wide variety of graphics equipment, and applications, which a communication system must support makes this seem a

* This paper is based on doctoral research conducted at Purdue University between January, 1971 and August, 1973. It resulted in a functioning system which allows thirty high speed terminals (storage tube graphics, refresh type graphics, alphanumeric CRT, remote batch stations, and remote computing systems) to communicate efficiently with a CDC 6500. Credit is due members of the Purdue University Computing Center staff, particularly John Steele, Jim Roberts, and Ross Garmoe, and Dick Garrett of Mechanical Engineering at Purdue for their contributions to the design objectives and details of this communication system.

** A more detailed rational for these constraints, as well as specific details of the communication system components outlined herein, is included in the doctoral thesis which resulted from this work.[4]

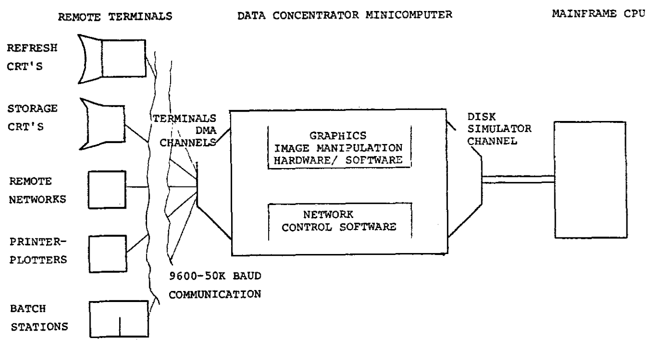


Figure 1—Data concentrator system components

difficult question. Consideration of each major equipment-application combination yields a surprisingly consistent answer.

Graphic images for output to storage-type terminals must naturally be produced and transmitted in an uncompressed form suitable for the vector generating circuits. Pictures such as labeled axes with a family of curves, perspective or orthographic parts layouts, or loading diagrams must be drawn in less than one second. If this drawing time is greater, the terminal user gets involved with the drawing process and loses track of the drawing's meaning and its place in his interactive design process.[3] Such images require between 600 and 1000 characters or a speed of up to 10,000 baud. (Baud is used as a synonym of bits per second in this paper although this is not technically correct for all communication schemes.)

Real time motion, for applications in which it is handled by the mainframe, also requires high speed communication. Dynamic CPU output should allow five to ten inch per second, relatively smooth motion requiring 10,000 to 50,000 baud depending on image complexity. In addition, the 10,000 baud speed is necessary for tracking input devices such as joysticks, tablets, and optical scanners.

Another requirement for usable interactive graphic systems is real time data acquisition capability. Multi-channel, ten bit resolution conversion systems also require from 10,000 to 50,000 baud depending on sample rate. Finally this 10K to 50K rate is supported by the need to load programs into minicomputer-complimented terminals from mainframe disks and cross assemblers and compilers. At these speeds, typical programs can be loaded is less than five seconds.

INTERFACE TO THE CPU

The connection between the central processor and data concentrator is the most critical in the communication system. The design of this connection must meet two severe constraints. First, it must support data rates of up to 150,000 (thirty terminals at 5000 cps) characters per second with minimum CPU overhead. Second, it must do that consistent with the input/output structure of the operating system to minimize modifications to that system.

The 150K data rate requires an entirely new approach to the CPU communication problem. Teletype timesharing systems generally support thirty-two to 128 terminals at ten to thirty characters per second generating less than three percent of this 150K rate. Such low speed systems can afford to interrupt the CPU on a character at a time basis, or an interrupt per character time (time division multiplexing) basis, or a line at a time basis. Increasing the number of interrupts by a factor of forty with any of those schemes virtually swamps the CPU.

It has been shown[4] that more than eighty percent of CPU interrupt service time goes to suspending the in-execution program (storing the registers, rolling the program to disk, rolling the interrupt processor in, . . .) and less than twenty percent to the actual data transfer. This suggests buffering transfers between the data concentrator and the mainframe into hundreds or perhaps thousands of character records before interrupting the CPU to transfer them.

Usually the most efficient system code in large scale computers is that which communicates with disk units. This is because disk communication is at the heart of the operating system operation and program execution. For this reason, the data concentrator should simulate a disk unit for each terminal connected to it. This disk simulator approach not only guarantees that all system and user code will be able to communicate with terminals with little or no system modification, but also insures that communication will be very efficient. The buffering requirement developed above is automatically met since disk transfers are by physical sector (between 320 and 1024 bytes for different mainframes).

If the data concentrator is to exactly simulate a disk unit for each terminal device, the concentrator must naturally be connected to a mainframe channel. Data rates between one half and one million bytes per second, which are characteristic of these channels, require concentrator controller hardware (not software) be used to simulate the disk controllers. This hardware must also have direct memory access (DMA) to the data concentrator in order to accommodate those very high data rates without swamping the minicomputer's CPU. Design of such controllers is a feasible but difficult task. In addition, the design is unique for each specific mainframe. But it is a well specified design problem since it must exactly simulate an existing piece of hardware, the disk controller.

The advantages of connecting directly to a mainframe channel are significant. First, this allows the kilobyte data rates necessary for driving dozens of 1000 to 5000 character per second (10,000 to 50,000 baud) graphics terminal devices. Second, the buffering into disk sector units decreases the CPU interrupt rate to less than required to drive an equivalent number of ten or thirty character per second terminals. Third, since mainframe disk controller specifications are constant for a given machine, any version of an operating system or any user program which can communicate with the disk units can communicate with the terminals. Summarizing, the disk simulator approach provides adequate speed for graphics devices while reducing CPU communications overhead and isolating the communications system from operating and programming system modifications and updates.

INTERFACES TO THE TERMINALS

Using a data concentrator minicomputer to make high speed graphics terminals look like disk units is nearly optimum from the mainframe CPU's point of view. The concentrator's cost can easily be justified in terms of reduced CPU overhead. (A suitable minicomputer is twenty to fifty times less expensive than a mainframe). It does, however, represent a significant investment in terms of purchase cost, programming, and maintenance liability, and must therefore be justified in terms of the number of terminals it can support.

The data concentrator can probably be justified if it costs less than ten percent of the value of all the terminals which it supports. Experience indicates that cost is between $40,000 and $75,000 for the minicomputer, mainframe channel interface, and terminal channel hardware. As a result, the concentrator must support between thirty and fifty low cost ($5,000 to $40,000) terminals. This is within current computer capabilities although care must be taken to balance the hardware- and software-performed functions to insure neither the minicomputer's processor, channel, nor memory time are swamped.

There are three major functions which must be performed by the terminal interface hardware. First, of course, it must transfer individual characters between data concentrator memory and the terminals. Since these are assumed to be remote terminals, transfer includes parallel to serial conversion meeting either synchronous or asynchronous communication specifications. Second, since terminal data will likely be ANSCII, characters must be converted to the mainframe's character set. In addition, line termination specifications of the mainframe must be met so that data is formatted exactly as on a real disk unit. Finally, transmission error detection and correction must be handled to insure reliable, long distance, high speed communication.

A critical requirement of the terminal interface hardware design is that all serialization specifications be variable under program control. These include character frame size (5 to 8 bits) transmission speed (110 to 50,000 baud), number of stop bits (1 or 2), parity type (even, odd, or none), and synchronous or asynchronous communication type. This allows any type of graphics device to be connected to the data lines, and varied dynamically during system operation, with its specification information determined and documented by the software.

In order for the concentrator processor to be able to handle the very high terminal data rates, each channel must have direct memory access (DMA). The thirty to fifty independent DMA ports and their necessary memory accessing, multiplexing, and control hardware make the terminal controller a very complex unit. It is, however, feasible to build into a suitably configured minicomputer for roughly $5000 to $10,000.[5]

DMA for characters going out to the terminals is a straightforward operation. A block of characters (perhaps a text line) is converted from the disk sector buffer into a temporary buffer area. Its initial address and a character count are passed to the channel, the characters are copied one by one to the terminal, and the processor is interrupted when the transfer is complete. For characters coming into the concentrator, however, the transfer is not count but end-character controlled. If the processor must look at each character for the terminator(s), the value of DMA is lost. Therefore, each channel must contain special character detect hardware which interrupts the processor when any of at least three different characters or sequences of the three are found.

The choice of three special characters is arbitrary but represents a minimum since the equivalents of RUBOUT, CARRIAGE RETURN, and INTERRUPT must be detectable for normal text input operations. Naturally, the character values should be variable under program control for each channel independently to insure maximum flexibility in terminals which can be supported. Allowing sequences of these three instead of any one simplifies meeting synchronous terminal specifications.

In addition to data transfer, the hardware must be designed to perform table-lookup character conversion and error detection in a one-step operation. Again, the requirement that these be done with hardware comes from the lack of processor time required to process characters on an individual basis with software. General experience with serial phone line communication indicates that error rates are low enough (one error in 10^5) to eliminate the need for automatic error correction schemes. Simple character parity, exclusive-OR block, and cyclic redundancy check block error detection can be used to trigger retransmission of bad blocks. These schemes should be optionally selectable per channel with the conversion hardware checking or producing the validation information.

A final note about terminal interface hardware is appropriate. For terminals located more than several miles from the data concentrator, modems are required with their associated disproportionately high (though decreasing) costs. For terminals inside this several mile radius, modems are not necessary. Line drivers can be built for less than $50 worth of hardware which operate on standard twisted pair wire at speeds up to 50,000 baud with very low error rates. Where appropriate, these line drivers can significantly reduce the per terminal costs associated with line interfacing.

DATA CONCENTRATOR SYSTEM SOFTWARE

As described in the preceding sections, data transfer is handled in blocks by the hardware controllers. The block size for transfers with the mainframe is determined by its disk sector size. Data within these sector blocks must be formatted to meet disk unit's line termination, end of record, and end of file specifications. The terminal block size should be variable by terminal channel to be most convenient for the devices connected to the concentrator. For standard alphanumeric communication, a block should contain one text line (i.e. terminated by a carriage return). For binary communication, a range of power-of-two sizes (32, 64, 128, 256) is generally most convenient for buffer allocation schemes.

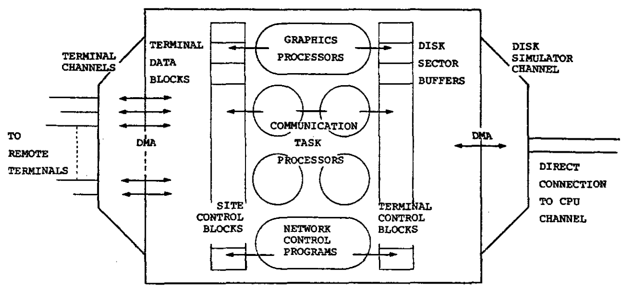


Figure 2—Data concentrator internal component relationships

The basic software communication task, therefore, involves accepting a block of data from one controller, validating and reformatting it into another block, and requesting another controller to transfer the new block to the next machine. This type of software can operate in either of two very different ways. Either way, the software operation is divided into tasks which are initiated by the completion of one hardware function (i.e. a block of data has been read into memory) and terminated by the initiation of the next hardware function (i.e. convert and validate the data block). The difference lies in whether the tasks have queues of terminal numbers which are waiting for that task or whether the terminals have queues of tasks. The latter has proven superior as it facilitates priority schemes for resource allocation during heavy loading for the higher data rate terminals.

Each physical terminal device (represented as a logical disk unit to the mainframe) must have associated with it a terminal control block (TCB) in the concentrator. This memory area (thirty-two sixteen bit words is likely adequate) contains the terminal dependent information such as special character codes and character conversion table address, and internal pointers into the disk sector blocks.

In addition to these TCB's, each communication line must have a similar site control block (SCB). This SCB contains the channel dependent information such as serialization specifications, error detection scheme, and pointers to the data blocks.

The mapping between terminal and site control blocks need not be in one-to-one correspondence. Many terminals (TCB's) may be multiplexed to a single line (SCB) via software, eliminating expensive multiplexing hardware while permitting maximum utilization of communication lines. Figure 2 shows the relationship between TCB's, SCB's, data blocks and the hardware controllers.

The dedication of two words in each control block provides an optimum means of queueing. The interrupt code, triggered by the completion of each hardware function (which separate software tasks as described above), need only store the address of the next task processor in one of these words. The processor monitor loop continually scans these queue words for each TCB and SCB. Upon finding one non-zero (indicating a request), the monitor loads the control block's initial address into a designated register and branches to the task processor. The processor, using the control block information as the parameters to a pseudo finite state machine, executes the task, perhaps modifies the control block, zeroes the requesting queue word, initiates the next hardware operation, and returns to the monitor loop. This scheme permits processor time for each terminal to be monitored and allows individual terminals or lines to be given priority. It also allows task processors to be added with virtually no modification to existing code.

GRAPHICS SUPPORT SYSTEM

The intent of the hardware controllers and software communication system is, of course, to support graphic terminals. Establishing a very high speed data path to each terminal with minimum mainframe CPU overhead is essential but not sufficient. To economically support large numbers of graphics devices, the CPU must be freed not only of communication overhead, but also of real-time graphic image manipulation. This can be done by moving image processing tasks into the minicomputers which accompany refresh-type terminals or into the data concentrator for storage-type terminals.

In order for programs to use processing capabilities other than of the mainframe CPU, the data concentrator must provide a communication path for passing subroutine calling parameters. This allows programs to be written as if for one machine but compiled into both the mainframe and the minicomputer. Naturally this is most effective if a single higher level language (i.e. FORTRAN) can be used requiring compilers for the minicomputer terminals, the concentrator, and the CPU.

Intermachine program execution is not essential, of course, but a minimum set of dynamic image manipulation capabilities should be handled outside the mainframe CPU. These include translation, rotation, and scaling of two dimensional images, the ability to read the location of any input device connected to the terminal, and dynamic display of a tracking cross or cursor which follows each input device. All should be initiated with a minimum set of control characters from the mainframe or from within the minicomputer.

Image manipulation tasks fit conveniently into the data concentrator queueing system described above. The alter and track operations can be independently coded tasks running reentrantly with the terminal control blocks providing storage for the current display parameters. A copy of the unaltered display image, as generated by the mainframe, must therefore be maintained in the concentrator. In this way, an interrupt from the mainframe or terminal causes the display parameters to be updated in the TCB and a task queued to apply their new values to the original image representation, transmitting the modified picture to the terminal as hardware move/draw commands.

The memory necessary for storing image representation for storage terminals demand a general discussion of data concentrator memory requirements. Experience indicates[4] that operating system task processors (including image processors), control blocks, and interrupt code require about 10,000 (16 bit) words. Communication buffer space is more difficult to estimate. Naturally, the larger the buffer size, the fewer CPU transfer interrupts (more disk sectors copied per interrupt), and the lower the CPU overhead. For high speed communication (10,000 to 50,000 baud), however, buffers are used for short periods of time allowing them to be timeshared very effectively and reducing the advantages of high core per terminal ratios. Again, experience indicates indicates that 24K to 32K is adequate for thirty to forty terminals. This leaves at least 20K (16 bit addressing usually permits 64K memories), for image representation. This is adequate for five to ten terminals although it increases the per terminal cost significantly.

The data concentrator task queueing structure facilitates one final terminal support option. Analog hardware is available (or can be built) at reasonable cost to perform three dimensional graphic image modification at significantly greater than real-time speeds. This suggests that such hardware be included in the data concentrator and thereby timeshared among several graphics terminals. Three dimensional manipulation tasks which have access to this hardware can easily be included in the software library operating in a reentrant mode on the TCB's just as do the two dimensional processors as discussed above. Thus, with a minimum hardware investment, low cost display terminals can have full three dimensional capabilities without burdening the mainframe CPU.

NETWORK POSSIBILITIES

Extensive attention has been directed over the past several years to computer networks. The concepts involved in networking are certainly worthy of consideration: economies of scale, load distribution, backup, variation in services, etc. Three major factors have, however, slowed the proliferation of computer networks. The first of these, and probably the most complex, comprises the political and economic problems involved in depending on or accepting responsibility for remote usage of very expensive mainframes.[6] The second factor involves availability and reliability of communication lines of sufficient speed to facilitate network communication. AT&T expects to offer 50,000 baud "Data Under Voice" service on a dial up basis to major cities beginning in late 1974, thus providing an apparently workable solution to this second problem. The third factor involves the inconsistencies between mainframes such as file formats, character sets, word lengths, compiler conventions, etc.

The data concentrator system discussed above provides a solution to the third factor, as shown in Figure 3. Each line from a distant network computer can be treated as any other multiplexed input, controlled by a site control block. Terminals requiring service of the local mainframe look exactly like local terminals through the linkage of that network SCB to several terminal control blocks. Character set, file format,

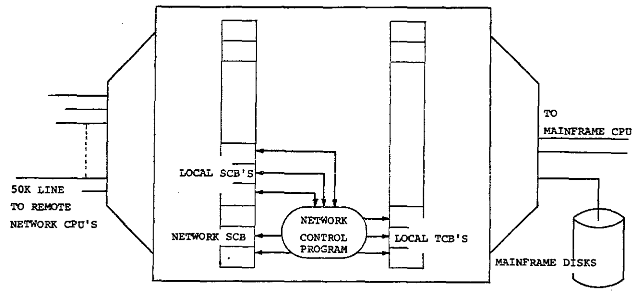


Figure 3—Network use of the data concentrator

and word length problems can easily be resolved by concentrator tasks using information about the remote mainframe contained in each TCB. In addition, local terminals can use the remote computer via tasks which link local SCB's to the multiplexed, network SCB.

The use of a data concentrator as a means of connecting local terminals to either a local mainframe CPU or to remote networks provides automatic backup capabilities. This is unlike many network schemes which depend on local CPU terminal interface hardware and therefore deny network access when the local CPU is down.

A reasonable improvement to the concentrator system involves hardware which allows the concentrator to communicate directly with local mainframe disk units. This is a straightforward addition. The concentrator channel must already contain disk simulation hardware (as discussed above), simplifying the physical connection. Since this direct disk connection is used only when the mainframe CPU is down, the software interlock problems normally encountered is dual access are eliminated. This permits, however, a terminal user to use a remote network computer with full access to his files while his own CPU is down.

CONCLUSIONS

This paper describes the considerations involved in the design of communication systems that permit economical high speed terminal use of large scale computers. The major requirements: minimum mainframe system modification and CPU overhead, 10,000 to 50,000 baud data rates for each terminal, flexible interface specifications, graphic image manipulation outside the mainframe CPU, and networking capability demand use of a data concentrator minicomputer. The hardware and software components of this concentrator are outlined as they pertain to the design requirements.

It must be emphasized that the disk simulator approach to the CPU interface virtually eliminates the need for software modifications to the mainframe operating or programming systems. It does require, however, relatively complex hardware to connect directly to a mainframe channel. Though complex, the hardware can often be built less expensively than manufacturer supplied interface units (i.e., IBM 270X or CDC 6671/6676 multiplexors) which can support only much slower, teletype-like devices and demand greater CPU processing overhead.

With the rapidly decreasing graphic terminal and modem costs, the time has come for timesharing communication systems which support high speed, image-oriented data transfer. In this paper at least one feasible approach has been outlined, involving some reconfiguration effort but without long term increases in communication system costs or CPU overhead. In addition to supporting graphics terminals, these improvements in communication systems are capable of supporting remote batch devices (card readers, line printers, electrostatic plotters, etc.) and capable of network communication with other large scale computers.

REFERENCES

1. Sutherland, I. E., "A Man-Machine Graphical Communication System," *AFIPS Conference Proceedings*, Volume 23, 1963.
2. Hopewell, L., "Trends in Data Communication," *Datamation*, August, 1973.
3. Gunn, M. A., *On the Development of Computer Graphic Design Tools for Enhancement of Creativity*, Ph.D. Thesis, Purdue University, 1974.
4. Boardman, T. L., *On the Exploitation of Computing Systems and Computer Graphics in the Development of Effective, Economical Engineering Design Processes*, Ph.D. Thesis, Purdue University, 1973.
5. *Modular Computing Systems*, Reference Manual MODCCMP III, February, 1971.
6. Williams, L. H., "A Functioning Computer Network for Higher Education in North Carolina," *AFIPS Conference Proceedings*, Volume 42, 1972.

APL as a development tool for special-purpose processors*

by SAMUEL LEVY, RICHARD H. DOYLE, and RALPH M. HELLER

IBM Corporation
Gaithersburg, Maryland

INTRODUCTION

APL[5] used as a programming language has been implemented in computing systems to provide time-sharing with remote terminals.[1,8] Since the operative principles guiding the design of APL include simplicity and practicality of application,[2] APL is extremely effective as a development aid for special purpose processors.

This paper describes two unrelated processor design projects which were significantly aided through the utilization of APL. The paper will emphasize the common features of many processor development tasks and show how APL may be applied advantageously.

The first example concerns an APL model which assisted in the conceptual development and evaluation of arithmetic algorithms for an associative processor whose arithmetic operations are based on residue arithmetic computations.[4] The second example is an APL simulation of a processor with subsequent use of this simulation to develop object code. The processor, together with this object code, served as a prototype for a special processor development task. In both examples, APL provided timely and meaningful assistance.

It is not within the scope of this paper to compare APL with discrete simulation languages. The GPSS and SIMSCRIPT family of languages[6] as well as APL program packages such as SUPERMOD[7] provide far more detailed and comprehensive analyses than we are addressing in this paper. Methods and advantages of using APL specifically as an interactive simulator generating system for small computers are discussed in Reference 10.

Some of the basic features of APL which make it attractive for our purposes are as follows: A repertoire of application-oriented APL functions can be built incrementally, with interactive testing performed at each stage of development. At first, these functions can be quite primitive, such as representing the behavior of a single processor component (e.g., a shift register). Later, as a hierarchy of functions are built, testing will become more complete. The advantage of interactive development is that interspersed development and testing can proceed with confidence at every stage until a preliminary processor model is available. The model can then be used for activities such as processor design, arithmetic algorithm development, and object code development. Then, characteristics of the conceptual processor may be modified and the APL model revised. Iteration of these steps will enable development and evaluation to proceed until project completion.

Other significant features of APL are: a relatively short program development cycle and the generality of the language notation. There is evidence[9] that it takes about three times as long to program and debug a problem using FORTRAN or PL/1 as it does using APL/360. Since APL operates in interpretive mode rather than through early binding by a compiler, program execution may be resumed immediately following editing of the line of the function which had terminated.

The APL notation is mathematically rigorous. Just a decade ago, a forerunner of this notation was used to present a precise formal description of a complete computer system, the IBM System/360.[3] The primitive APL functions, expressible as distinct symbols and the APL operators provide a concise convenient notation for handling algorithms, with the added advantage of direct execution in the notation. APL accrues additional advantages from its simplicity, versatility and flexibility. These are properties of the language itself which are familiar to APL users, and so, are not emphasized here. Their presence will be obvious as the examples are unfolded.

RAAP MODEL AND ALGORITHM DEVELOPMENT

The RAAP (residue arithmetic associative processor) is an associative processor having a number of semi-independently operating associative memories, and whose basic arithmetic operations are performed in residue arithmetic format.[11] The processor design is not fixed and the work reported on herein concerns preliminary development work on basic RAAP memory structure, and algorithm development and evaluation.

The RAAP consists of a group of associative arrays operating in parallel under the control of one program which resides in the control store (refer to Figure 1). Each associative array is associated with a different modulus of the residue

* This work was supported in part by the Air Force Office of Scientific Research on Contract No. F44620-73-C-004.

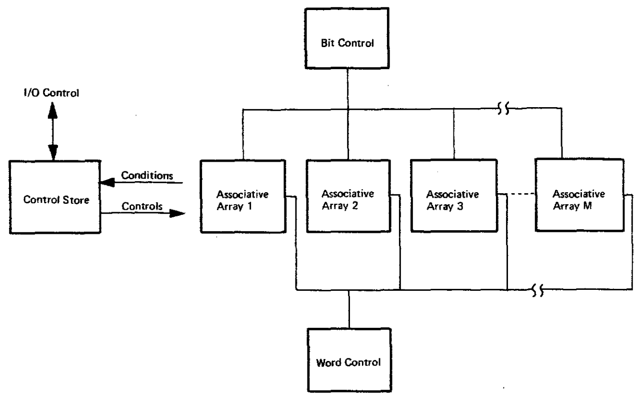


Figure 1—Block diagram of associative processor

base representation of a data word. One microcoded program simultaneously controls all residue moduli computations (associative memory operations).

Although no attempt was made to finalize the architecture or design of a specific RAAP configuration, an approach to processor organization functioning and control was needed for the development and evaluation of the arithmetic algorithms. Figure 2 is a general block diagram illustrating present thinking.

For conceptual and practical convenience we consider the RAAP to consist of a number, N, of basic associative memory units, A1, A2, . . . , AN, having a similar form. Each associative memory unit contains a control field, and from one to four fields to store data. The conventional mask, data and selector registers are associated with each memory.

The associative memories are controlled by one master control unit which implements the arithmetic and input/output conversion algorithms. A key feature is that the associative memories can interact through their selector registers. The memories are interconnected through a Selector Transfer Register which permits the transfer of selector register contents from one memory unit to another.

Memories A1 and A2 have respective storage fields F1 and F2. These memories will handle input/output to the

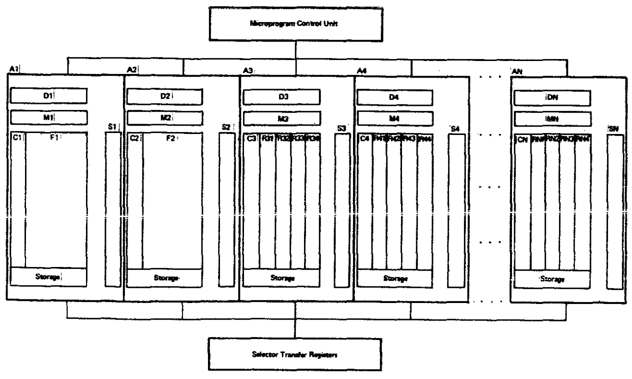


Figure 2—RAAP structure

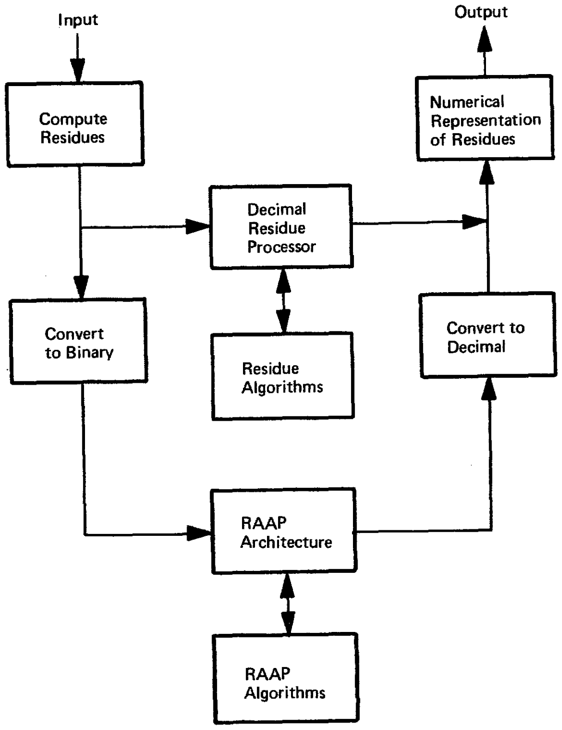


Figure 3—RAAP model data flow

RAAP and are also involved in the input/output conversion to residue base representation. Each of the associative arrays, A3, A4, . . . , AN, handles one modulus of the residue base representation. As indicated in Figure 1, array AK has control field CK, residue fields RK1, RK2, . . . , RK4, and data, mask and selector registers, DK, MK, and SK, respectively. Each memory in Figure 2 has an area labeled storage associated with it. This represents the portion of the memory which contains the necessary constants for residue conversion, overflow computations, masking and other operations.

Our study was confined to problems of processing positive integers only, there being no loss in generality for multiplication. Each associative array was assumed to contain up to four data fields and one field of control bits. Since residue base representations require significantly fewer bits per operating field than standard representation, the application of residue arithmetic in an associative processor might be expected to provide a processing speed advantage over a conventional associative processor. An APL model was developed to aid in investigating required arithmetic algorithms, processor architecture, and RAAP processing speeds as compared to conventional associative processing.

The APL model consists of two segments representing both a decimal residue processor and a binary RAAP model as illustrated in Figure 3. Input data consists of a vector of positive integers. The model converts the input data to a matrix containing the residue values corresponding to each integer. The decimal residue processor is simply a hypothetical processor which serves as a vehicle for preliminary examination of the properties of residue arithmetic and convenience in output display. The residue matrix is transformed into a three-dimensional binary array which serves as input to one data field of the RAAP architecture. After arithmetic processing by the RAAP, the contents of a data field are transformed into a matrix of residues. The outputs from the RAAP or the decimal residue processor are then converted to numerical representation through the use of the Chinese Remainder Theorem. The output appears as a vector of positive integers.

In the APL model (refer to Figure 4) an associative array is represented by four data fields (FLD1-FLD4) and a group of control bits (CTL). The number of control bits varies with the algorithm under test. All associative arrays are under the control of the SEARCH and WRITE functions.

In an associative processor a search operation interrogates specified bits (usually denoted by a mask register) of all words in the memory and identifies to a selector register, those words which identically compare with the contents of a data register. In this model, the right argument of a search statement specifies the following:

1. Bit mask—four data fields and M control bits
2. Selector register—one of two selector registers with/without logical OR capability
3. Data register—content associated with bit mask.

In an associative processor, a write operation transfers the contents of a data register to specified bits (based on the mask register) of those memory words identified by the content of a selector register. In this model, the right argument of a write statement specifies the following:

1. Bit mask—three data fields and M control bits
2. Selector register—one of two registers with true or complement form
3. Data register—content associated with bit mask.

One of the basic problems in developing a computer model for a conceptual processor configuration is the requirement that the model be readily adaptable to change. The dynamic linkage of APL functions considerably alleviates the necessity for recoding call sequences. For example, if a variable NAMES contains a name list of APL functions and a variable FLAG contains a vector of indices to NAMES, then an APL function INVOKE can be defined to invoke a call sequence based on FLAG. Figure 5 illustrates this example. The execute function is used by INVOKE to perform the construction and execution of statements under program control.

```
      NAMES

END
INPUT
INITIAL
OUTPUT
START

      FLAG
5 3 2 4

      FLAG INVOKE NAMES

    THIS WILL PRODUCE THE FOLLOWING CALL SEQUENCE:

START
INITIAL
INPUT
OUTPUT
END
```

Figure 5—INVOKE example

The structure of the RAAP model is illustrated in Figure 6. One may run the model by assigning the indices to FLAG and executing the function RESIDUE. The function RESIDUE initializes the model and INVOKE dispatches each of the major functions. Names starting with the letter "B" apply to functions which operate on binary data and names starting with the letter "P" apply to functions which printout data. TRACE lists the names of all functions invoked.

The basic operation of the model is illustrated in Table I. The contents of the variable FLAG show the call sequence. Run number 8 of RESIDUE produces a listing of all functions invoked, the input data and its residue representation,

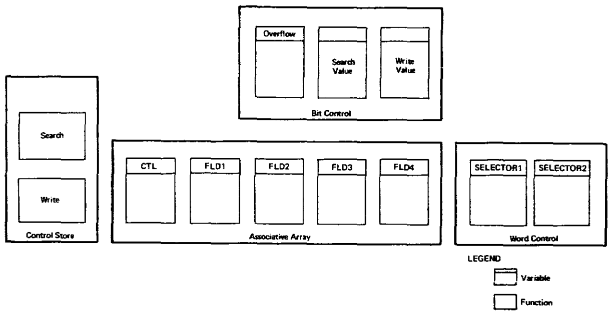


Figure 4—Associative array model

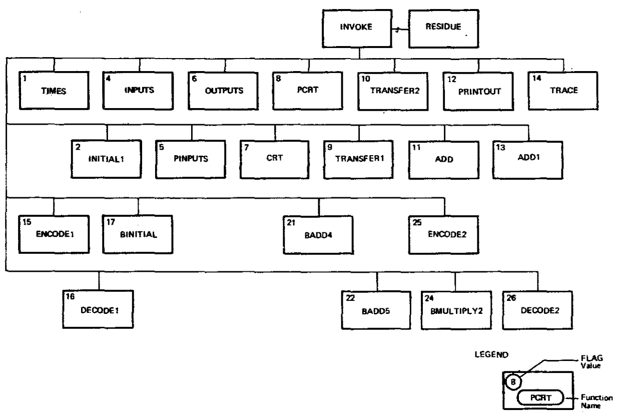


Figure 6—Program structure

TABLE I—Illustration of Basic Operation

```
      FLAG
2 14 4 5 10 11 6 8 12

      RESIDUE 8

INITIAL1
TRACE
INPUTS
PINPUTS
TRANSFER2
ADD
OUTPUTS
PCRT
PRINTOUT

INPUTS INPUTS1

INPUT       RES(63)  RES(61)  RES(59)  RES(55)  RES(53)  RES(47)  RES(43)  RES(41)

        70     7        9       11       15       17       23       27       29
       105    42       44       46       50       52       11       19       23
       500    59       12       28        5       23       30       27        8
 564950500    16       37       12       15       21       19       31       20
1969887318    42       44       38        3       13        6       42        6
 940422545    62        2       10       40       32       15       32       10

PCRT        MINVERSE MINVERSE2

OUTPUT      RES(63)  RES(61)  RES(59)  RES(55)  RES(53)  RES(47)  RES(43)  RES(41)

       140    14       18       22       30       34       46       11       17
       210    21       27       33       45       51       22       38        5
      1000    55       24       56       10       56       13       11       16
1129901000    32       13       24       30       42       38       19       40
3939774636    21       27       17        6       26       12       41       12
1880845090    61        4       20       25       11       30       21       20
```

the residue representation of the output data, and the numerical value of the output data. Each of the six input words was converted to eight residue values. For example, an input of 500 is represented by 27 with respect to a modulus of 43. The contents of data field 1 were transferred to data field 2 by the function TRANSFER2 and the function ADD added the number to itself. For example, 27+27 with respect to modulus 43 is 11. Taking into account all eight moduli, the output value is 1000 which indeed is 500 doubled.

The results of binary multiplication are shown in Table II for run number 23. One input (1473) was converted to residues with respect to six moduli. The contents of data field 1 were transferred to data field 2 and the function ENCODE2 converted these values to binary form. The algorithm under test, BMULTIPLY2, initialized FLD2 with the multiplicand and FLD3 with the multiplier (which are equal). The results of performing binary multiplication in the RAAP are shown under output. For example, 42 times 42 with respect to modulus 53 is 15. Each residue was then converted back to its decimal equivalent by the function DECODE1 and the function CRT (Chinese Remainder Theorem) computed the value 2169729 from the six residues. Thus, since 1473 squared is equal to 2169729 the run was successful.

In Reference 4 much more elaborate examples of APL utilization for RAAP structure and algorithm development are presented. The interactive use of the model with printout of binary field content during intermediate arithmetic stages permitted the quick isolation of deficiencies. Considerations of carry and overflow techniques, field utilization, modulus adjustment, selector transfer and control bit usage became evident as successive versions of the algorithms were tested. In this manner several addition and multiplication algorithms were developed.

TABLE II—Illustration of Binary Multiplication

```
      FLAG
2 14 4 5 10 12 17 25 24 16 6 8 12

      RESIDUE 23

INITIAL1
TRACE
INPUTS
PINPUTS
TRANSFER2
PRINTOUT
BINITIAL
ENCODE2
BMULTIPLY2
DECODE1
OUTPUTS
PCRT
PRINTOUT


INPUTS INPUTS1


 INPUT RES(64) RES(63) RES(61) RES(59) RES(55) RES(53)

 1473     1       24       9       57      43       42

INITIAL CONDITIONS
⍋FLD2:
 0 0 0 0 0 1   (mod 64)

 0 1 1 0 0 0   (mod 63)

 0 0 1 0 0 1   (mod 61)

 1 1 1 0 0 1   (mod 59)

 1 0 1 0 1 1   (mod 55)

 1 0 1 0 1 0   (mod 53)
⍋FLD3:
 0 0 0 0 0 1

 0 1 1 0 0 0

 0 0 1 0 0 1

 1 1 1 0 0 1

 1 0 1 0 1 1

 1 0 1 0 1 0

(1 2 3 4 5 6)

BMULTIPLY2 BTRANSFER31 BTRANSFER12A BASICADD4 BASICADD5 BTRANSFER13 SEARCH2

AVERAGE INSTRUCTIONS PER BIT
SEARCH=  99.8   WRITE=  91.5   TOTAL=  191.3

PCRT      MINVERSE  MINVERSE2

OUTPUT   RES(64)  RES(63)  RES(61)  RES(59)  RES(55)  RES(53)

2169729     1        9        20       4        34       15
```

POL PROCESSOR EMULATION AND INTERPRETER DEVELOPMENT

This example involves the functional simulation of a processor and the subsequent development of an interpreter to enable the processor to directly execute problem-oriented language (POL) statements. The purpose of this project

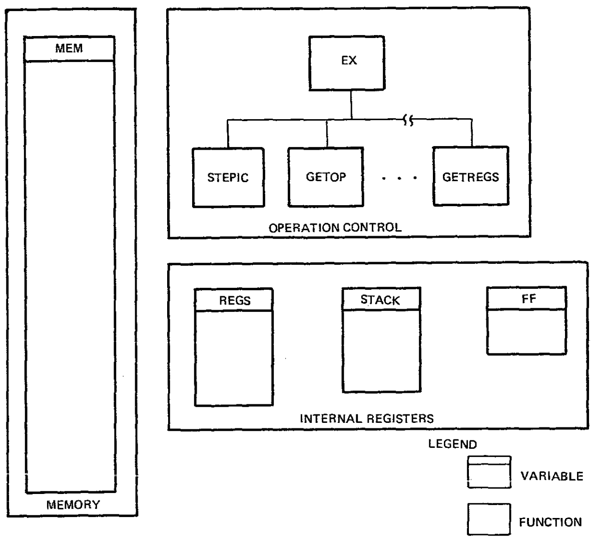


Figure 7—POL simulation

was to build a feasibility model of a proposed low-cost, stand-alone machine which would interpret and directly execute problem-oriented language code. As a stand-alone device, this proposed machine would be required to support only one user at any given time, and hence, high execution speed was not a primary factor in its design. Accordingly, the design approach taken was to emulate, via software, the interpretation and execution of the source-language code, rather than to build special-purpose microcoded hardware.

For the prototype, a small commercially available micro-computer was ordered. At that time, no support software existed to assist in the assembly or debugging of code written for this machine. For this reason, as well as a delivery lead time in excess of six months, a decision was made to emulate the POL processor with APL as the medium.

The first task was to create, in an APL workspace, a functional replica of the set of object machine registers and the memory. This included general registers (REGS), a "push down" stack of address registers (STACK) and flip flops (FF) to indicate the result of arithmetic operations and parity (refer to Figure 7). These were easily represented as APL Boolean arrays, (vectors and matrices). APL routines to perform elementary machine functions such as STEPIC (STEP Instruction Counter), GETOP (GET OPeration code), etc., were then coded in APL, and a main control routine was written to call the other routines, treating them as subroutines. Each of the approximately 50 instructions of the object machine repertoire was representable within this main routine by not more than three short lines of APL code, most by one line.

As the process of building this package progressed, it was possible at all stages to check proper emulated processor operation by entering sample machine code into the simulated memory, limiting the subset of machine code to that which was then possible to interpret. Interaction was of particular value during this process, since program errors were eliminated en route, and not allowed to go undetected until the package was thought to be complete. The package was supplemented by APL support software that facilitated loading, trace during execution, and code error diagnostics of object machine code.

The final task consisted of writing the machine object code that was to interpret and execute the problem-oriented language as entered from a keyboard. Using the APL terminal, both in its normal role of user-APL interface, and in the role of a simulated prototype keyboard for machine code entry, and again, later, for problem-oriented language source-code entry, the prototype package was developed and entirely checked out through processor emulation. Again, the interactive feature of APL was exploited, this time with machine source code developed and tested incrementally. At every stage of code development, trial runs were invoked, and each partial package was thoroughly checked out before proceeding with the next development phase.

It is estimated that, counting both tasks, (i.e., creating the machine simulation tools and coding the product machine code), more than 150 "turnarounds" were invoked. The effective total turnaround time, measured in human terms, (i.e., human time spent unproductively waiting for results), amounted to no more than 10 to 15 minutes.

CONCLUSIONS

In this paper we have illustrated ways in which APL can be useful as a development tool for special-purpose processors. It provides simplicity and practicality of application in a time-sharing environment and, as such, provides a powerful capability for processor modeling.

In the first example, the APL model was extremely useful during the study of algorithms for a residue arithmetic associative processor. It proved quite flexible for conceptual processor features such as: bit precision per residue, number of associative memories, modulus per associative memory, number of control bits per memory word, number of words per memory and bit size per I/O word. Dynamic reconfiguration of the APL model aided in reducing program debug time as well as enabling interactive development and analysis of the arithmetic algorithms. The APL model enabled prediction of timing characteristics for the arithmetic algorithms operating in the associative processor.

For the second example, an APL simulation of a micro-computer enabled development of an interpreter in machine language code and validated the machine specifications. APL was then successfully used as an emulator for the prototype processor. The entire effort was accomplished in three weeks, and the developed software ran faultlessly in the micro-computer on the first and all subsequent trials.

ACKNOWLEDGMENT

The authors wish to acknowledge the support and encouragement of Col. R. Ives and Col. T. Wachowski of the Air Force Office of Scientific Research and Dr. R. Van Blerkom of IBM.

REFERENCES

1. Falkoff, A. D. and K. E. Iverson, *APL/360 Users Manual*, IBM Corporation, (GH20-0683-1) 1970.
2. Falkoff, A. D. and K. E. Iverson, "The Design of APL," *IBM Journal of Research and Development*, pp. 324-333, Vol. 17, No. 4, July, 1973.
3. Falkoff, A. D., K. E. Iverson and E. H. Sussenguth, "A Formal Description of System/360," *IBM Systems Journal*, pp. 197-261, Volume Three, Numbers Two and Three, 1964.
4. Heller, R. M. and S. Levy, *A Study of Arithmetic Algorithms for a Residue Arithmetic Associative Processor*, Tech. Report on AFOSR Contract No. F44620-73-C-004, September 20, 1973.
5. Iverson, K. E., *A Programming Language*, Wiley, New York, 1962.
6. Kay, I. M., "An Over-the-Shoulder Look at Discrete Simulation Languages," pp. 791-793, *AFIPS Conference Proceedings*, Vol. 40, 1972.
7. Leiner, A. L., *SUPERMOD: An Analytic Tool for Modeling the Performance of Large-Scale Systems*. IBM T. J. Watson Research Center Report No. RC2796, February 12, 1970.
8. Pakin, S., *APL/360 Reference Manual*, Science Research Associates, Inc., Chicago, 1968.
9. Streeter, D. N., "Cost Benefit Evaluation of Scientific Computing Services," *IBM Systems Journal*, pp. 226-231, Volume Eleven, Number Three, 1972.
10. Brame J. L. and C. V. Ramamoorthy, "An Interactive Simulator Generating System for Small Computers," pp. 425-449, *AFIPS Conference Proceedings*, Vol. 58, 1971.
11. Szabo, M. S. and Tanaka, R. I., *Residue Arithmetic and its Applications to Computer Technology*, McGraw-Hill Book Co., 1967.

Narrowing the generation gap between virtual machines and minicomputers

by HENRY J. THEBERGE and ERIC E. BEAVERSTOCK

Honeywell Information Systems Inc.
Waltham, Massachusetts

INTRODUCTION

A virtual machine system has been defined by R. Goldberg[3] as "A system . . . which . . . is a hardware-software duplicate of a real existing machine in which a non-trivial subset of the virtual machine's instructions execute directly on the host machine, . . .".

Large computer systems have been shown to be excellent environments on which virtual machine systems operate.[1,2] If these systems were implemented on small computer systems ("minicomputers"), a desirable environment for software development would exist because of the capability of evaluating and testing different operating systems. Minicomputer applications such as graphics, process control, intelligent terminals, could be developed on a virtual machine while sharing support equipment. Current mini-computer inadequacies prevent the successful implementation of virtual machine systems. These inadequacies include:

- no automatic memory protection
- the lack of sensitive instruction recognition
- no virtual memory facilities
- no virtual I/O handling

This paper presents a method for implementing a virtual machine system in a mini-computer. The approach circumvents existing inadequacies by utilizing a software/firmware-implementable, interpretive, high-level, system programming language. The system, if implemented in software, has an emulator rather than a virtual machine monitor as the term has been currently defined.[5]

This paper is divided into three parts

- Description of the higher level language
- Description of the Emulator
- Description of the Virtual Machine Monitor

DESCRIPTION OF THE HIGHER LEVEL LANGUAGE

The language we considered is TRAIL.[4] The basic elements of TRAIL are:

1. a source language
2. an intermediate interpreted language
3. an interpreter

Source language

The source language was designed to facilitate system design and debug phases, to enhance documentation, and to allow greater productivity of programming. The language facilitates recursive programming and elicits a programming discipline for the separation of logical flow of control from the basic manipulative operation. TRAIL effectivity separates conditional tests and branching decisions (rules) from straight-line code (action sequences). Each program is called a graph and is comprised of rules and action sequences. An action sequence is a free statement consisting of a sequence of action calls, variable references, and action operators without any *Go To* or conditional instructions. Action sequences apply to a run time work-stack and may be nested to any depth. The work-stack is used mainly for expression evaluation and parameter passing.

Variables are allocated dynamically at graph-call time and deallocated at graph-exit time. Variables can access virtual addresses since TRAIL allows definition of a virtual memory mapping scheme.

These features provide greater productivity in design and debug phases, and enhance communication between programmers via simplified documentation procedures.

Intermediate interpreted language

The source language is converted into an intermediate or target language by a translation process. The target language is interpreted at execution time. This interpretive approach achieves portability, and programs are treated as relocatable data by the interpreter. Each action sequence is encoded into a string of byte sized intermediate language operators for interpretation. This choice of an interpreted target language accomplishes minimum object code size (roughly 50 percent smaller than assembled code) and machine independence of the developed software.

Interpreter

The interpreter is either written in assembly language or firmware implemented. It is designed to maximize speed

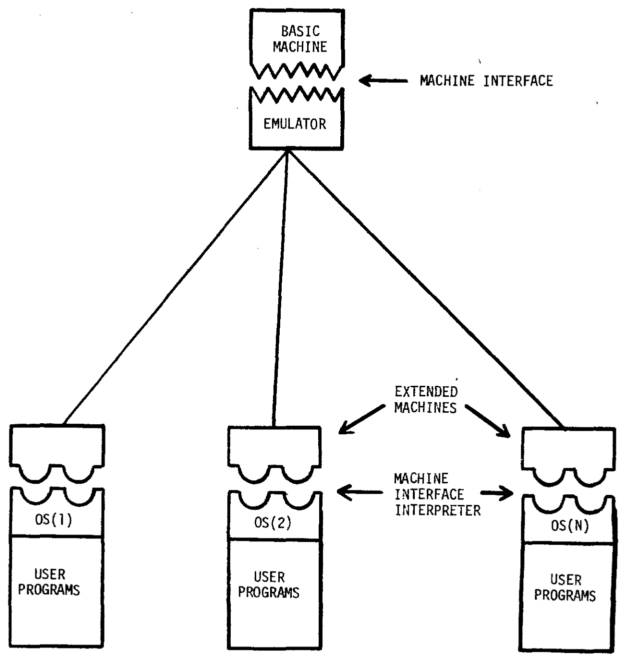


Figure 1

with respect to interpreting the byte sized string of intermediate language operators, and at the same time minimizes the actual code size required. When a graph is entered, the interpreter allocates relative space for an environment. The environment consists of the work stack and an area for the interpreter to maintain pertinent run-time information. By associating a unique environment with each graph or groups of graphs, the concept of coroutine is introduced. A coroutine is a graph which can relinquish control to another coroutine and later be reactivated to continue computation. All environment management is controlled by the interpreter. The interpreter I/O mechanism is activated, when an I/O request is identified. The mechanism performs a type of subroutine call to the I/O controller, which performs the desired I/O action.

These features of TRAIL along with others [4], provide an excellent base for the creation and support of an emulator or a virtual machine monitor for a small machine.

DESCRIPTION OF THE EMULATOR

Figure 1 depicts an emulator system through the use of language primitives as designed by Robert P. Goldberg of HIS.[1] The architectural design consists of a basic machine that executes an emulator. The emulator is written in assembly code and performs all I/O operations, memory mapping functions, and other virtual machine requirements. The interpreters, also written in assembly code, interpret the TRAIL code of the various operating systems and user programs. An interpreter is executed directly by the basic machine and the interpreter passes control to the emulator after having interpreted an action operator requiring some virtual machine operation i.e. space allocation. The emulator, having performed the necessary operations, determines which interpreter is to be activated. The interpreter transfers control between the operating system and the user programs by means of the coroutine mechanism as previously described.

The emulator as proposed here, solves many problems that currently exist in implementing virtual machine concepts on mini-computers. Automatic memory protection is provided by the emulator because it controls the memory allocated to each interpreter and its programs. The interpreter(s) recognizes certain action operators to be sensitive—i.e., an instruction requiring some virtual machine function—thus the sensitive instruction is dealt with at the interpretive level. A virtual memory scheme is implemented within the emulator, and is activated by the emulator when required. Virtual I/O handling is also provided by the emulator.

Thus, with some software design modification, one can easily move toward emulation on a mini-computer.

DESCRIPTION OF THE VIRTUAL MACHINE MONITOR

Figure 2 depicts a virtual machine monitor system. The basic model requires that an interpreter be *firmware* (microcoded) implemented. The virtual machine monitor is written in the language being interpreted (e.g. TRAIL). The operating system, also written in the same interpreted language, controls the execution of the various user programs. The

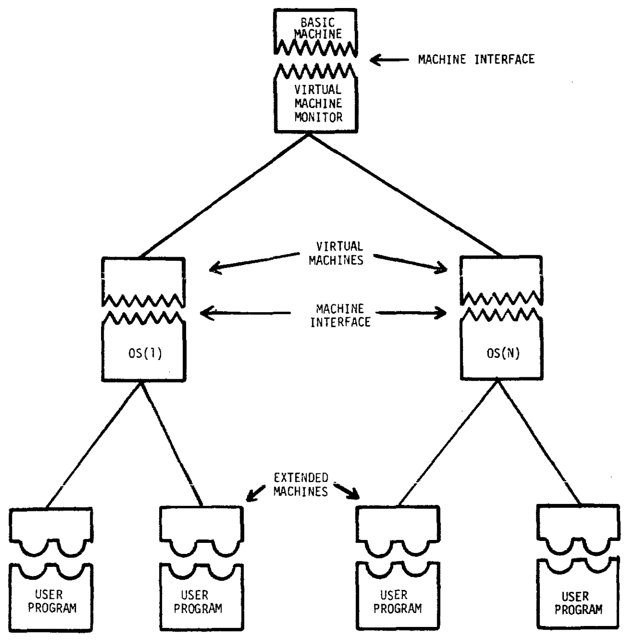


Figure 2

firmware transfers control between the operating system, user programs, and the virtual machine monitor by means of a coroutine mechanism. The firmware passes control to the virtual machine monitor when virtual machine functions are required. An important characteristic is that the virtual machine monitor is not performing an instruction by instruction interpretation of the operating system or user programs, but rather remains inactive until called upon by the firmware.

The virtual machine monitor accomplishes memory protection by gaining control from the firmware as required by the various programs. Certain action operators which require virtual machine functions are recognized and the virtual machine monitor is activated by the firmware. Virtual memory, is also handled by the virtual machine monitor when given control through the firmware. The virtual machine monitor accomplishes the mapping of the set of resources found in the virtual machine configuration into the set of resources existing in the basic machine.

This virtual machine concept provides for the creation of various extended machines upon which user programs may run. Since the virtual machine monitor is written in TRAIL, monitoring and debugging is readily accomplished.

CONCLUSIONS

From the described configurations, the problem of implementing a virtual machine system on small machines is reduced. The emulator approach enhances current capabilities of mini-computers without major modifications or extensions; however, a slow down in execution time will result. This problem can be avoided if the virtual machine monitor approach is taken.

The simplicity and compactness of the TRAIL language makes it possible to implement the virtual machine requirements on a mini-computer.

Although TRAIL was mentioned as the interpretive language to be used, it is noted that any interpretive language with the same basic characteristics can be utilized. With either of these approaches, a broader use and greater reduction of the limitations of mini-computers are accomplished.

BIBLIOGRAPHY

Gagliardi, U. O. and R. P. Goldberg, "Virtualizable Architectures," *Proceedings of the 1972 ACM International Computing Symposium*, Venice, Italy, April, 1972 pp. 527-538.

Goldberg, R. P., "Architectural Principles for Virtual Computer Systems," *Report #24-72, Center for Research in Computer Technology*, Harvard University, Cambridge, Mass.

Goldberg, R. P., "Virtual Machines-Semantics and Examples," *IEEE Comput. Soc. Conf.*, Boston, Mass. Sept. 1971 pp. 141-142.

Lechner, R. J. and W. Stallings, "A Minisystem Programming Language," *Proceedings of the 1973 ACM Annual Conference*, Atlanta, Georgia, (August 1973), pp. 174-182.

Lichstein, H. A., "When Should you Emulate?," *Datamation 15*, 11, Nov. 1969, pp. 205-210.

Pipelining—The generalized concept and sequencing strategies*

by C. V. RAMAMOORTHY and K. H. KIM

University of California
Berkeley, California

INTRODUCTION

In this paper, we generalize the concept of pipelining to increase throughput, processing speed, resource utilization and reliability. Its wide application spectrum is demonstrated and major design problems such as sequencing, reconfiguration, etc., are indicated. A scheme called the dynamic sequencing and segmentation model (DSSM) is proposed as a solution for providing efficient sequencing with very low overhead. The model is analyzed under various realistic environments and its performance is evaluated.

Pipelining can be defined as a technique of imbedding concurrency in a computer system by implementing it in the form of a *pipeline*, a configuration of independent autonomous units each of which is dedicated to perform a specific subfunction in an overlapped mode with others. An autonomous unit or a segment of a pipeline is also called a *pipeline-segment* or a *facility-segment*. Pipelining has emerged as an important aspect of computer architecture especially of scientifically oriented computers. A large number of computers have been built with one or more pipelined functional units.[1,2,3,11]

Pipelining, however, still remains as an ad hoc procedure and a specialized technique of exploiting computational parallelism. Total parallel processability in computations has not been fully exploited in conventional pipeline systems. In this discussion, we classify the approaches to parallelism exploitation into two categories: (1) a *passive approach*, and (2) an *active approach*. In the passive approach the parallelism is exploited without any modification of execution sequences within a program, while in the active approach the exploitation is achieved by the automatic detection of the parallelism inherent in the program at various levels[7,9] and the judicious sequencing of these detected parallel processable tasks. Conventional pipelining has mostly been confined to the passive approach to exploiting the local parallelism, i.e., the parallelism between adjacent tasks.

Pipelining in a functional unit is achieved in two steps: (1) segmenting the functional unit into a number of facilities (segments) each requiring generally the same amount of execution time and (2) streaming independent jobs through these segments.

Levels of pipelining

The principle of overlapped concurrent operation can be employed effectively at various levels in the computer architecture.

Pipelining at the gate level is exemplified in the design of the instruction processing unit (IPU). The IPU can be decomposed into various functional segments, namely, instruction fetch, instruction decode, address generation, etc. (Figure 1). It takes one minor cycle for a task (instruction) to pass through each segment. Thus after a stream of tasks enters this pipeline, the pipeline starts outputting one task per minor cycle. Microprogram prefetch, that is, overlap of decoding the current microinstruction with fetching the next microinstruction is another example at this level.

The next level for the application of pipelining is the subsystem level, with the pipelined arithmetic units[3,4] being the typical examples. Pipelined ADD, MULTIPLY, DIVIDE[3] and SQUARE-ROOT[6] functions have been in existence in a number of contemporary computer structures. Figure 2 is the conceptual representation of the operation of the DIVIDE unit of IBM 360/91, where, as D_i iteratively approaches 1, N_i approaches N/D, the quotient.

Generalization

So far, the design practices have confined the principles of pipelining (overlapped concurrent operations) to facility-segments with almost equal processing times and whose control and data flows are linear or unidirectional. Since computational sequences cannot always be divided into equal processing time-segments and since recursion and iteration are common computational modes, it would be desirable to remove the restrictions and extend the precept of pipelining. We shall next discuss the concomitants of this generalization. We shall characterize the generalized pipelining into the following aspects.

The first criteria would be *segment-lengths* (processing times) of each facility-segment in the pipeline. These processing times can be the same or *variable*. Multiprogramming

* Research sponsored by the National Science Foundation, Grant GJ-35839, and the U.S. Army Research Office—Durham, Grant DA-ARO D-31-124-73-G157.

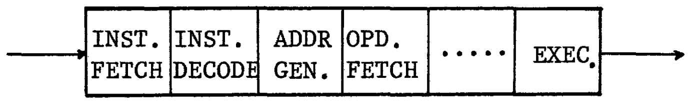


Figure 1—The pipelined IPU

is an example of a pipeline with two variable-length segments (CPU and I/O).

The second factor for characterizing pipelines is the *direction of control and data flow*. It could be *unidirectional* (e.g., the IPU) or *bi-flow* or *bidirectional* (e.g., IBM 360/91 DIVIDE unit).

The third criteria would be degree of *replication* of the pipeline. There may be several pipelines which perform identical operations on different operands (e.g., TI-ASC's arithmetic pipes).

The fourth characterization of the pipeline system is its *reconfigurability*. The system could be non-reconfigurable (e.g., IBM 360/91) or dynamically reconfigurable. Figure 3 shows an example of a replicated reconfigurable pipeline as implemented in the TI-ASC computer.[5] There are four identical pipelined arithmetic units (AU's), each consisting of eight segments. All AU's would perform the same operation. Four segments among eight ones in each AU can be configured into a pipeline configuration for FIXED POINT MULTIPLY at one time while at another time six of them could be configured into a FLOATING POINT ADD pipeline. Although examples show linear pipelines, configurations in general need not be restricted to be linear but could be planar (two-dimensional) or multi-dimensional.

The facility-segment may be not only hardware but also hardware-software complex of any complexity. The latter type can be often seen in the pipeline of the system level (Figure 4). A number of special-purpose computer networks can be also looked upon as this class of pipelining.

The development of various pipeline systems corresponding to combinations of these characteristics is rather an evolutionary step in the design of pipeline systems. The fully generalized concept of pipelining encompasses both conventional parallel processing and pipelining as subsets. Further improvement of resource utilization is achievable. The fully generalized pipelining is capable of utilizing resources which would have been idle in the conventional approach. In the case of a pipeline which is reconfigurable in multi-dimensions, those resources could often be configured into a certain pipeline and used for computations resulting in the increased utilization. Improvement of reliability is another gain of generalization. Reconfigurability and replication in the generalized pipeline provide the sound basis for the incorporation of graceful degradation as well as fault diagnosis and recovery techniques. An example of the generalized pipeline, with dynamic reconfiguration and replication is shown in Figure 5.

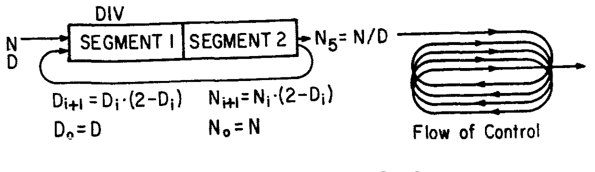


Figure 2—IBM 360/91 divide

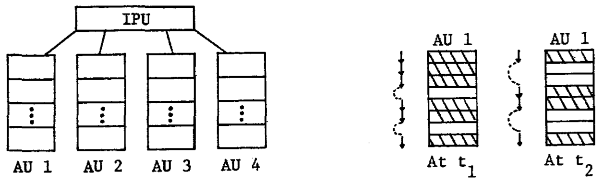


Figure 3—The replicated dynamic pipeline (e.g., AU of TIASC)

Design problems

Now that generalization of the concept has been discussed, we shall consider problems involved in designing such a pipeline system. Some of the common problems in the design of pipeline systems are the following: (1) design configuration of pipelines, (2) sequencing strategy, (3) resolution of storage conflicts, (4) determination of program suitability, and (5) efficient execution of sequences (efficient utilization of resources).

The design configuration of a pipeline system involves the determination of proper segment-lengths, their processing speeds, the degrees of their replication and their reconfigurability to achieve the necessary performance criteria (throughput rate) on a spectrum of jobs.

The basic problem in sequencing is to decide the order of executions of independent tasks such that their completion time is minimized. The performance of a pipeline system is highly sensitive to sequencing strategies. Figure 6 illustrates the effect of sequencing in the case of a simple pipeline consisting of two segments. The sequencing problem is in general twofold: (1) the development of the algorithms and (2) the overhead generated in using it. In other words, one has to develop an efficient algorithm[10,12,14] such that the overall execution time including the overhead is minimized. There is an obvious trade-off here since the closeness to optimality of the algorithm increases its run-time overhead. In the next section, an effective approach is introduced.

Another design problem is concerned with storage conflicts. This problem generally occurs when more than one task being executed concurrently needs to access the same memory module. The memory conflicts can be classified into two types. The first type is the *local conflict* occurring between consecutive instructions in a program when they are executed concurrently and need to access the same memory module. The other type is the *global conflict* oc-

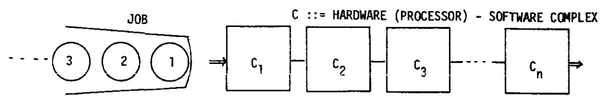


Figure 4—Pipelining at the system level

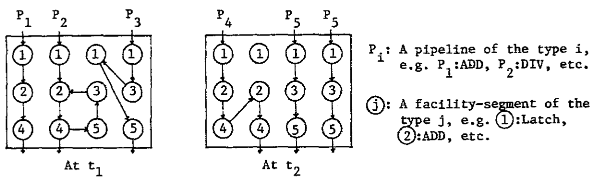

P1 P2 P3
P4 P5 P5
At t1
At t2
Pi: A pipeline of the type i, e.g. P1:ADD, P2:DIV, etc.
(j): A facility-segment of the type j, e.g. (1):Latch, (2):ADD, etc.

Figure 5—An example of the generalized pipeline

curring between independent tasks which are executed concurrently. Techniques have been devised in many contemporary pipeline systems to reduce the local storage conflict. One typical example of the solution is memory-interleaving based on the principle of program locality. Since in general, global conflicts occur in addition to local conflicts there is a need for new techniques for reducing memory interferences.

The next design problem is the one of program suitability. Some programs may reconfigure the pipeline excessively without utilizing each configuration effectively. It would be desirable to develop the techniques for restructuring such ill-formed programs so that the restructured ones may become more suitable (reduce total execution time) for the specific type of pipelining (Figure 7).

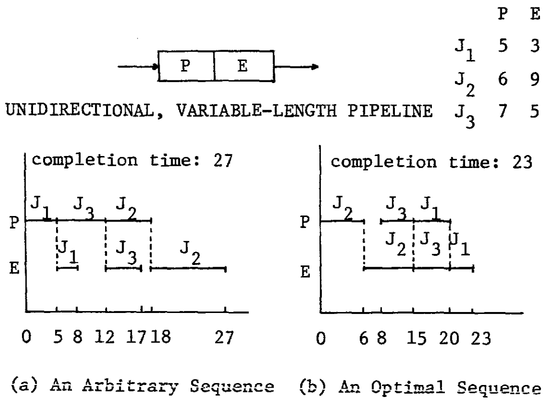
Efficient execution of sequenced tasks is another design problem. Efficiency consideration arises in the process of activation and synchronization of tasks. These bookkeeping operations must be minimized in both extent and frequency.

In the rest of the paper, one of these common problems, namely the sequencing problem is discussed in detail with a description of a new approach of resolution.

SEQUENCING STRATEGIES

Concept of the dynamic sequencing and segmentation model (DSSM)

As mentioned before, an efficient sequencing strategy is essential to the pipeline system, but its run-time overhead



	P	E
J_1	5	3
J_2	6	9
J_3	7	5

Figure 6—An example of the effect of the sequencing

```
K = A * B                                K = A * B
N = B / C                                P = D * E
L = K + 7                                R = G * H
P = D * E                                N = B / C
O = N + 5          12                    S = R / I            3
R = G * H                                L = K + 7
M = L + 3   reconfigurations             M = L + 3   reconfigurations
S = R / I       required                 O = N + 5       required
Q = P + F                                Q = P + F
K = M * Q                                T = S + J
T = S + J                                P = Q + T
N = M * O                                K = M * Q
P = Q + T                                N = M * O

(a)  An Ill-formed Version               (b)  A Restructured Version
```

Figure 7—An example of program restructuring

becomes expensive. A run-time overhead is accumulated in concurrent processing since a number of decisions regarding resource allocation, etc. must be made dynamically. Actually this has been the main obstacle in parallelism exploitation.

To overcome this problem a scheme called the dynamic sequencing and segmentation model (DSSM) is developed which overlaps the unproductive overhead (bookkeeping and administrative) computations with the execution of computational tasks so that the effect of overhead is greatly diminished and an efficient exploitation of parallelism is realized.

In this system a dedicated unit for sequencing, called the *sequencer*, operates on one segment of the computational job while the processor, called the *executer*, executes the previous (already sequenced) segment (Figure 8).

The DDSM itself is an example of pipelining at the high (system) level. For the purpose of sequencing and segmentation, computational jobs are modelled by the *parallel task graph* (PTG).[9] A *task* is a set of instructions which, once initiated, can be carried out to its completion by a pipeline without the need for additional inputs. Thus one can consider a single instruction a task. A task can be often partitioned into a set of *subtasks*, each of which can be executed by one pipeline-segment. In the rest of the paper, no distinction is made in use between a task and a subtask. A program or a job in the pipeline system consists of a set of tasks. A PTG is a loop-free directed graph in which (1) each node represents either a task or a set of tasks in a loop, and (2) a transition exists from the node i to the node j if and only if the task j depends for its initiation on a result generated by the task i.

In the PTG, each loop or a strongly connected subgraph is abstracted into a single node, or task. A sequencing procedure is developed to execute the loop-free PTG. If any loop or strongly connected subgraph is large (many nodes and/or large execution times), its execution sequence during

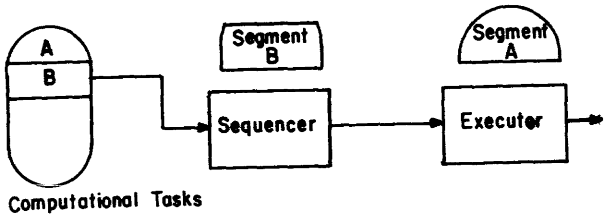


Figure 8—The concept of the DSSM

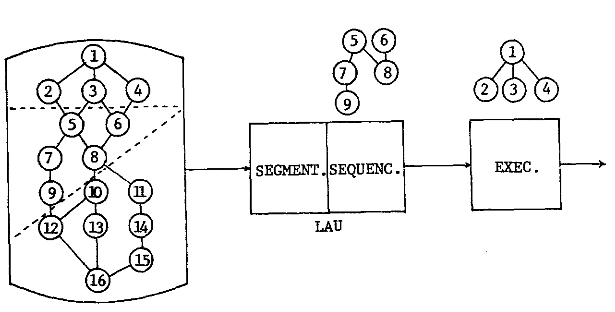


Figure 9—An example of a PTG in execution

each iteration can be analyzed by the segmentor and sequencer of the DSSM, and the sequencing information is provided to the executer. This is equivalent to opening up the loops so that they are in a spiral sequence of tasks repeated a finite number of times. If the loop or the strongly connected subgraph of tasks is small it can be considered as a single task. The segmentor and the sequencer together may be called the *look-ahead unit* (LAU) in the sense that both sequencing and segmentation are look-ahead functions (Figure 9).

Design principles of the DSSM

Dynamic sequencing

The sequencer determines the execution sequence of a segment of tasks while a previous segment of tasks are being executed. This sequencing technique possesses high adaptability to dynamically varying environments, which is superior to the static (pre-) sequencing. Presequencing or static sequencing may result in inefficient executions because it cannot take into account the variation of job characteristics over time, or the run-time changes in resource utilization patterns due to external interrupts, etc.

Basically, the sequencing problem can be formulated as follows. Given the configuration and characteristics of the pipeline, and a set of related tasks to be executed, it is required to develop an execution sequence (schedule) that reduces the total execution of the set of tasks. It is assumed that the task dependencies and the task execution times (or their estimates) are known. In this paper, we shall focus ourselves to the problem of scheduling and sequencing a set of related tasks in a single large program as represented by its PTG. The problem of sequencing unrelated and independent jobs is a simple case and we shall not consider this. It is well-known that the overhead for computing the optimum sequence increases almost exponentially with the size and the structure of the task graph. Moreover, the resulting sequence could at best be suboptimal because of variances in their predicted values of execution times during run-time, etc. A heuristic algorithm which produces a reasonably efficient sequence with much less overhead would be highly desirable. The heuristic should accomplish the following. It should update the set of ready tasks according to the precedence relationships and the availability of facilities and it should select a task from the ready set by applying various selection rules. One example of a selection rule is to calculate the length of the longest path in time from each of the ready nodes (tasks) to the exit node and then to select the node with the longest path among them.[14] Since the suitability of any heuristic is highly dependent upon the system configuration and the characteristics of input jobs, comparative evaluation of algorithms under the given environment will be the reasonable approach for selecting a suitable algorithm. An example of the sequenced PTG is presented in Figure 10. Figure 10(a) shows an example of the PTG where each node (task) is associated with the necessary information for sequencing, the estimated time and the kind of facility required for the execution of the node (task). A double line in Figure 10(b) denotes the determined sequence of the parallel processable tasks competing for the same kind of facility. For instance, nodes 3 and 6 represent two parallel processable tasks requiring the same kind of facility, F_1 and they are sequenced into the execution order (3,6). The execution of the program according to this sequence is expected to take 105 time-units.

Dynamic segmentation

Segmentation here is also a dynamic segmentation analogous to the dynamic sequencing. It behaves according to the distribution of the work-load in the DSSM which is a dynamically varying characteristic.

Segmentation consists mainly of two parts: (1) To decide on the size of the next computation-segment, i.e., the number of nodes in the next segment to be sent to the sequencer, and (2) To select the nodes in the given task graph for that segment. The efficient segmentor has to pick up the appropriate size of segment for sequencing such that the sequencing time of the new segment is almost the same as the execution time of the current segment. Once a segment is selected, its execution time can be estimated and this is used to determine the size of the next segment. Therefore, it is desirable to know the nature of variation of overhead (time to generate an efficient execution sequence) with the size (number of nodes) of the segment analyzed (i.e., overhead characteristic).

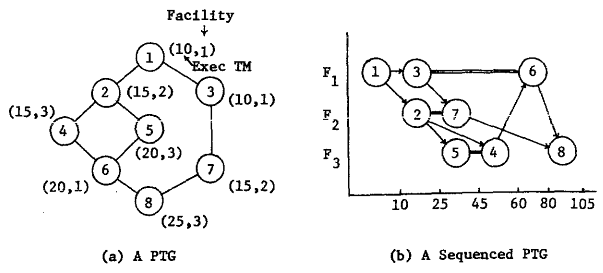


Figure 10—An example of the sequenced PTG

The other factors that affect the overhead have been ignored for simplicity. Once the algoirithm is fixed, this relation can be obtained by experiments and utilized in the DSSM.

Figure 11 shows an example of the overhead characteristic for a specific sequencing discipline. The curve represents a portion of the data obtained by the experiment with the randomly generated PTG's (refer to later section). The number of nodes contained in each PTG ranged between 4 and 120. We used the heuristic algorithm developed in Reference 10 for sequencing. The selection rule in this algorithm is basically to calculate the priority for each task i in the set of ready tasks by using the following priority function $p(i)$ and then select the task of the highest priority.

$$p(i) = \mathrm{SIGN}(T(i) - T_s(i)) \cdot \mathrm{MIN}(T(i), T_s(i))$$

where $T_s(i) = \mathrm{MAX}_{j \in S(i)}[T(j)]$, $S(i)$ = a set of successor tasks of the task i, and $T(i)$ = execution time of the task i. The simulated DSSM was run on a CDC 6400 computer. One time-unit of overhead in Figure 11 is roughly equal to 1 millisecond taken by the simulated sequencer for analysis. The smooth curve in Figure 11 is an interpolated approximation of the data represented by the discontinuous line shown in it. If a certain segment is going to take 75 time-units for execution, the suitable size of the next segment will be of 21 nodes. Of importance here are the shapes of the curves rather than their numerical significance, since the latter depends upon the method of implementation. The curve approximates a polynomial. The reason for this is that the analysis time of a segment grows rapidly with the number of tasks (nodes) in it.

Next, the picking up the given sized segment from the task graph, is done by maintaining precedence relations between the tasks. A sophisticated method requiring a large overhead would not be favorable unless it makes the significant increase in the system performance. This appeared to be valid by experiments with two algorithms of different degrees of sophistication.[8] One of them called the *precedence*

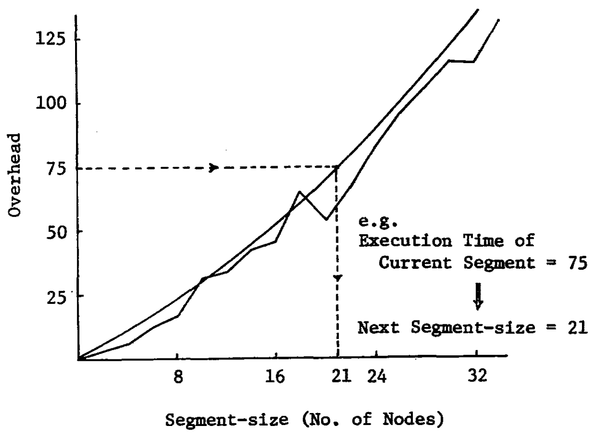
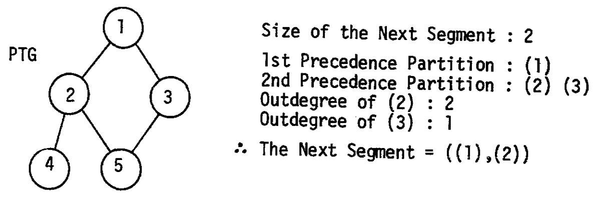


Figure 11—The overhead characteristic curve



PTG
1
2
3
4
5
Size of the Next Segment : 2
1st Precedence Partition : (1)
2nd Precedence Partition : (2) (3)
Outdegree of (2) : 2
Outdegree of (3) : 1
∴ The Next Segment = ((1),(2))

Figure 12—The PPO algorithm

partition and outdegree (PPO) algorithm is shown below for the purpose of illustration. The principle is to select nodes in the order of precedence partitions and in the order of outdegrees within the same partition (Figure 12).

The other algorithm used in the experiments was the *precedence partition and random assignment* (PPR) algorithm which is different from the PPO algorithm only in that nodes within the same precedence partition are selected randomly. For the example PTG in Figure 12, this algorithm will randomly pick up either nodes (1) and (2) or nodes (1) and (3) as the next segment.

Balancing the workload

As was discussed in the preceding section, the method of selecting the segment to be sequenced should be carefully designed since the effectiveness of the DSSM is highly sensitive to it. When the execution time of the segment just analyzed is accurately known, the sequencer would select a new segment whose analysis time (overhead) is equal to the execution of the segment (just analyzed). If this state of affairs is sustained, that is, the execution time of the current segment is almost equal to the sequencing time of the next segment, we shall call this the *normal* or *balanced state*. The proper size of the next segment is chosen by consulting the overhead curve (Figure 11).

On the other hand, when the execution times are not accurately known, then either the sequencer or the executer may occasionally have to wait for the other to complete its function. When there is a fixed amount of buffer storage between the sequencer and the executer, the segment already analyzed by the sequencer may be queued until the executer becomes available. If the execution of the current segment finishes before the sequencer completes its analysis of the next segment, the executer would be idle until the next segment is ready. In this case, the DSSM is said to be in the *sequencer-overloaded state*. On the other hand, when the executer is busy processing a segment, either the next segment just analyzed by the sequencer has to be queued or the sequencer would have to wait if there is not space left in buffer storage. If there is at least one segment queued in the buffer storage when the sequencer just completes the analysis of the next segment, the DSSM is said to be in the *executer-overloaded state*. In any of these cases, the performance of the DSSM may become somewhat degraded.

Many factors are ignored in the evaluation of the overhead.

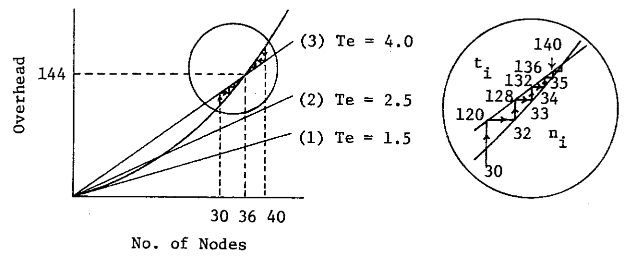


Figure 13—Analysis of the overhead characteristic curve

So, there exists a possibility that the workload in the DSSM may be unbalanced sometimes. In such a case the workload can be balanced dynamically in the following simple manner. Suppose the DSSM is in the executer-overloaded state. This situation is detected by the sequencer examining the status of the buffer storage each time it completes the analysis of a segment. Then the segmentor will select a larger segment of the size corresponding to the overhead equal to the sum of execution times of all segments queued up (including the segment just analyzed) so that the sequencer may complete the analysis of the next segment about the time when the executer completes executions of the segments queued. In the case of the sequencer-overloaded state, the following two cases can be considered. If the execution time of the segment just analyzed is accurately known, the segmentor takes a segment of the size corresponding to the overhead equal to the execution time of the analyzed segment, called a *segment of the proper size.* On the other hand, when the execution time is not accurately known but roughly estimated, it is presumed that the sequencer-overloaded state may persist if the segmentor takes a segment of the proper size. Thus the segmentor takes a segment of the size smaller than the regular size. This is then repeated until the workload distribution reaches the balanced state. In this way, the DSSM can resume the balanced state from any unbalanced state.

Behavioral analysis

By analyzing the overhead curve, two interesting properties of the DSSM can be discovered.

First, the stabilized segment-size is predictable. That is, the size of the computation-segment (number of nodes or tasks) becomes almost fixed after some transient time. We shall call this the *stabilized segment-size.*

The straight lines 1, 2, 3 of Figure 13 called the *normal execution time curves* represent the expected execution time of the segment depending upon the segment-size, when T_e is 1.5, 2.5 and 4.0 respectively. Here T_e denotes the *average effective execution time per node* in the given PTG and is obtained as follows:

$$T_e=\frac{T_a}{\text{ADOP}},$$

where T_a denotes the *average execution time per node* and ADOP denotes the *average degree of overlapped processing* which is in turn defined as the *average number of nodes being executed concurrently* during the execution of the given PTG. More specifically, T_a is the average amount of time required for executing one task (node) by the required facility in the executer. It can be interpreted as the quantitative measure of the level of a task which is directly proportional to the level of a pipeline. Provided that the executer is always busy, $1/T_e$ is the average number of nodes which the DSSM executes in a unit of time. T_e is dependent upon the level of a task and the ADOP which are dependent upon the job-characteristic and the system configuration.

We shall call the crosspoint of the normal execution time curve with the overhead curve (excluding the origin) the *stabilized point.* Provided that the overhead curve has a monotonically increasing property, there will exist a unique stabilized point. For instance, the stabilized segment-size is 36 when $T_e=4.0$ in the diagram. The reason is as follows. For the sake of explanation, let n_i denote the number of nodes in the ith segment picked up by the segmentor and t_i denote the normal execution time corresponding to the size of the ith segment. Suppose n_1 is 30 which is smaller than the stabilized size 36 (Figure 13). Then $t_1=n_1\times T_e=30\times 4.0=120$. Now n_2 is determined from the point in the overhead curve corresponding to t_1 $(=120)$ time-units of overhead. So, $n_2=32$ and $t_2=n_2\times T_e=128$. Similarly, $(n_3=33, t_3=132)\Rightarrow(n_4=34, t_4=136)\Rightarrow(n_5=35, t_5=140)\Rightarrow(n_6=36, t_6=144)\Rightarrow(n_7=36, t_7=144)\Rightarrow\cdots$. By the same reasoning, it can be intuitively seen that n_i iteratively converges to the stabilized size 36 in the case where n_1 is greater than 36.

Next we can determine the *lower bound of the average execution time per node* (T_a) in the DSSM. The line 1 of Figure 13 is the derivative of the overhead curve which passes through the origin. It corresponds to $T_e=1.5$. This T_e multiplied by the ADOP which is constant, is the lower bound of T_a. Below this bound the execution of a job in the DSSM could result in the completion time longer than the one by the sequential execution without any segmentation and sequencing. Since T_e is dependent upon the level of a task, there is a practical limit to the level at which the DSSM can be effectively applied. One approach to overcome this limit would be to reduce the overhead by the specialized implementation of the critical portion of the sequencer using firmware or hardware.

Performance evaluation and implementation

In this section, we present some results of performance evaluation studies carried out to validate the feasibility of the DSSM for implementation. This evaluation was done by simulation.

A PTG is the main input to the simulated DSSM and the connectivity matrix[9] was adopted for the internal representation of the PTG.

In order to test a wide range of inputs, the random PTG generator (RPTGG) has been developed. It produces the directed graph of random connection without loops, and the user can control the arc density and the pattern of the graph in order to tune it up to the desirable pattern based on intuition or experience. It is a useful tool for various studies in parallel processing. The basic algorithm is shown in Figure 14.

The overhead characteristics used was the one determined by the experiment in an earlier section of this paper. The execution time of each task was randomly generated using the average value T_a. Simulations were performed for several values of T_a (1, 2, 5 and 10 time-units). One time-unit represents 1 millisecond taken by the simulated executer for execution. The whole simulator was prepared in FORTRAN and run on the CDC 6400 computer.

The first observation was the negative effect of the sophisticated segmentation algorithm. The benchmark algorithms used were the PPO and the PPR algorithms introduced earlier. By simulation, the PPO algorithm didn't show any significant improvement of the gain compared to the PPR algorithm while its overhead was many times larger than that for the PPR algorithm. The PPO algorithm was discarded and only the PPR algorithm was used throughout the remaining simulation.

As a measure of performance, we used the following quantity, the gain by overlap.

$$\text{Gain}(\%) = \left(1 - \frac{T_p}{T_s}\right)*100$$

where T_p = Total completion time for the program in the pipeline system and T_s = Total completion time for the program in the sequential (nonoverlapped) system.

The following diagrams show the typical results obtained (Figures 15(a), 15(b)).

Curves 1 and 2 represent the case where a job is sequenced as a whole without being segmented. That is, the system doesn't have the segmentor. A whole PTG is analyzed at a time by the sequencer and then sent to the executer. Curve 1 is the case where the overhead is completely ignored, while curve 2 contains the full overhead. Therefore, curves 1 and 2 are the upper and lower bounds of the gain obtainable by the system without the segmentor.

Curves 3 and 4 represent the case of the DSSM. Curve 3

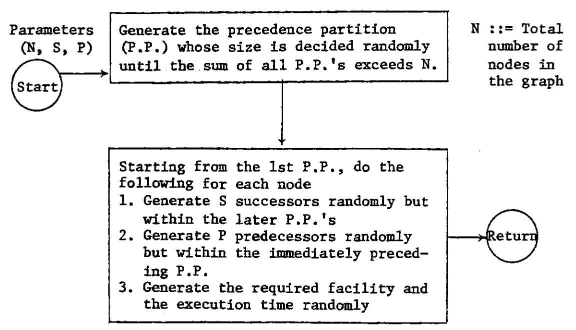

Parameters (N, S, P)
Start
Generate the precedence partition (P.P.) whose size is decided randomly until the sum of all P.P.'s exceeds N.
N ::= Total number of nodes in the graph
Starting from the 1st P.P., do the following for each node
1. Generate S successors randomly but within the later P.P.'s
2. Generate P predecessors randomly but within the immediately preceding P.P.
3. Generate the required facility and the execution time randomly
Return

Figure 14—The random parallel task graph generator

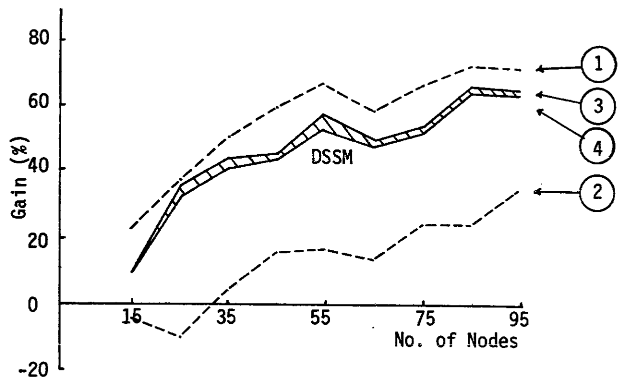


Figure 15(a)—Performance of the simulated DSSM when $T_a = 10$

is the case where the overhead required for segmenting and sequencing the first segment, is completely ignored, while curve 4 contains the full amount of the overhead.

Curves 1 and 3 can be regarded as those situations in which jobs are continuously entering the system and the sequencing of the new job is fully overlapped with the execution of its preceding jobs.

On the other hand, curves 2 and 4 can be regarded as those in which jobs are entering at discrete intervals and the sequencing of the new job is not at all overlapped with the execution of preceding jobs. Therefore, the gain under the continuous operation will range between those of curves 1 and 2 for the system without the segmentor and between those of curves 3 and 4 for the DSSM.

It is noteworthy that the DSSM maintains a high gain close to the maximum upper bound of performance (curve 1). This implies that it is highly adaptive to a dynamically varying environment. This can be more precisely stated by using the following notations. $G_i(n)$ denotes the gain represented by the curve i in the diagram corresponding to n nodes. $AG(n)$ denotes the *average gain obtained by the DSSM* in the case of n nodes. $AG(n)$ is obtained from the diagram as $\frac{1}{2}\cdot[G_2(n)+G_3(n)]$. N denotes a set of all instances of the

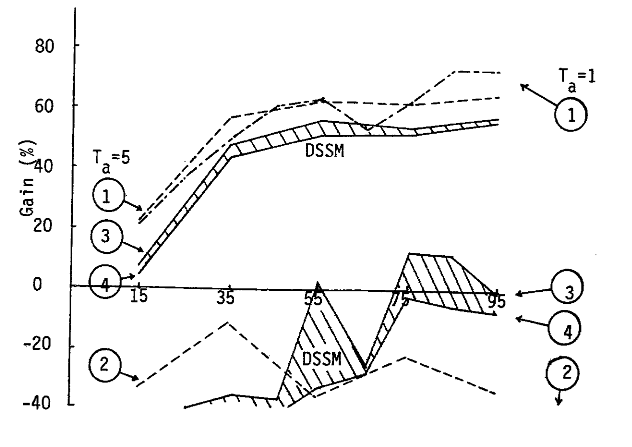


Figure 15(b)—Performance of the Simulated DSSM when $T_a = 1,5$

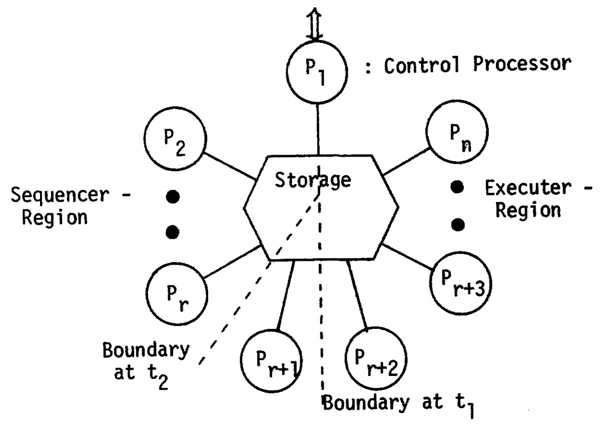


Figure 16—Reconfigurability in a pipelined computer network

discrete variable n which represents the number of nodes. Then, the *efficiency of the dynamic sequencing and segmentation*, Q, can be defined as follows:

$$Q::=\underset{n\in N}{\mathrm{AVG}}\left[\frac{\text{Average gain obtained by the DSSM}}{\text{Maximum upper bound of the gain obtainable}}\right]$$

$$=\underset{n\in N}{\mathrm{AVG}}\left[\frac{AG(n)}{G_1(n)}\right]=\left[\sum_{n\in N}\left(\frac{AG(n)}{G_1(n)}\right)\right]\Big/\#(N)$$

where $\mathrm{AVG}^{n\in N}[X(n)]$ represents the average value of $X(n)$'s corresponding to all $n\in N$, and $\#(N)$ represents the cardinality of N. As indicated by the diagram, the values of Q obtained from simulations were mostly over 85% where T_a was larger than 2 time-units.

As expected from the analysis of the overhead curve, the simulation results showed poor performance where the average execution time per node was less than the lower bound (when $T_a=1$ in Fig. 15(b)).

In short, simulation confirmed the validity of the analysis as well as the steady high performance of the DSSM.

The implementation of the DSSM must be based on the characteristics of the environment. The level of pipelining, the capability of multiprogramming and the multiplicity of facilities are the prime factors which affect the implementation. Also the level of language in which the input tasks are prepared is one of the factors which affect the method and time of parallelism detection. It is much simpler to detect parallelism from the machine language program than from the high level language program. In the case of the high level language input, the implementation varies depending upon whether the program is compiled and executed or directly interpreted.

The physical capacity of the LAU consisting of the sequencer and the segmentor may be much smaller than the one of the executer because (1) the algorithms employed for the sequencing and the segmentation normally don't make use of any complex operations and (2) the input data to this unit is an abstract model of the program instead of the program itself. It could be a special purpose processor with a small storage whose critical functions are hardware- or firmware-implemented.

The executer may vary from a set of small processors to a set of large scale processors.

Extension of the DSSM

Extensibility of LAU

The sequencing and the segmentation are not the only functions which can be provided by the LAU. In other words, the LAU can be easily extended to perform other desirable look-ahead functions. Typical examples are the look-ahead for branching decisions and the fault diagnosis and recovery. It is often possible that the execution of one segment provides enough information to make branching decision in advance before the decision point is encountered in later segments. Also the LAU can periodically diagnose the part of the executer without interfering with the execution. In addition, it can assist dynamic storage allocation.

Explorations of various useful look-ahead functions and the relevant implementation techniques are under way.

Pipelined computer network

Concepts of the pipelining and the DSSM can be effectively applied to a reconfigurable computer network which consists of a number of processors. The processors in the network can be partitioned into two sets according to their functions. One set acts as sequencers while the other set acts as executers. Since there can be several sequencers, more than one sequencer may be employed to the sequencing of several segments of a single job (this mode of sequencing is called the *parallel sequencing*) or a set of independent jobs.

The ratio of the number of executers to the number of sequencers is variable and thus the system becomes highly adaptive to the varying workload distribution. In other words, each processor may function as a sequencer at one time and as an executer at another time. The function of each processor is determined depending on the workload distribution in such a way that the system keeps the balance (Figure 16).

Amongst the executers, each processor functions as a facility-segment of a reconfigurable pipeline. Consequently, a set of facilities available for configuring pipelines are dynamically changing. New techniques must be developed to solve problems in pipeline configuration, sequencing algorithm, etc.

Because of the consistent decreases in the hardware cost, a computer network becomes feasible and attractive in the future, especially in the form of miniprocessor networks where each miniprocessor takes a specialized functional role, possibly via microprogramming.

CONCLUSION

The generalized pipelining appears to be a promising approach to further increase the throughput rate, processing speed and system reliability. A need exists for efficient implementation techniques. Pipelined computer network is a further generalization and needs to be investigated in detail.

REFERENCES

1. Thornton, J. E., "Parallel Operation in the Control Data 6600," *AFIPS FJCC*, Part II, 1964, pp. 33-40.
2. Anderson, D. W., et al., "The System/360 Model 91: Machine Philosophy and Instruction Handling," *IBM Journal*, Jan. 1967, pp. 8-24.
3. Anderson, S. F., et al., "The System/360 Model 91 Floating-Point Execution Unit," *IBM Journal*, Jan. 1967, pp. 34-53.
4. Hintz, R. G. and D. P. Tate, "Control Data Star-100 Processor Design," *Proc. COMPCON 1972*, pp. 1-4.
5. Watson, W. J., "The TIASC—A Highly Modular and Flexible Super Computer Architecture," *AFIPS FJCC*, 1972, pp. 221-228.
6. Ramamoorthy, C. V., J. R. Goodman, and K. H. Kim, "Some Properties of Iterative Square-Rooting Methods Using High Speed Multiplication," *IEEE Trans. on Comp.*, Aug. 1972, pp. 837-847.
7. Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," *Computing Survey*, Mar. 1973, pp. 31-80.
8. Ramamoorthy, C. V. and K. H. Kim, "Dynamic Sequencing and Segmentation in the Generalized Pipelining System," *Proc. 2nd Texas Conf. on Computing System*, Nov. 1973.
9. Ramamoorthy, C. V. and M. J. Gonzalez, "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs," *AFIPS FJCC*, 1969, pp. 1-15.
10. Reddi, S. S. and C. V. Ramamoorthy, *Sequencing Strategies in Pipeline Computer Systems*, Technical Report No. 134, Electronics Research Center, The University of Texas at Austin.
11. Chen, T. C., "Unconventional Superspeed Computer System," *AFIPS SJCC*, 1971, pp. 365-371.
12. Ramamoorthy, C. V. and H. F. Li, "Efficiency in Generalized Pipeline Networks," *Proc. NCC*, 1974.
13. Shar, L. E., *Design and Scheduling of Statically Configured Pipelines*, Ph.D. Thesis, Dept. of E.E., Stanford University, 1972.
14. Chandy, K. M. and J. R. Dickson, "Scheduling Identical Processors in a Stochastic Environment," *Proc. COMPCON 1972*, pp. 171-174.

Interfacing communication network to IBM System/360 and System/370 host processors—An end users viewpoint

by JOHN T. M. PRYKE and LAWRENCE J. MUNINI

Wang Computer Services
Tewksbury, Massachusetts

IBM COMMUNICATIONS EQUIPMENT

Before discussing interfaces per se, it seems wise to review briefly the existing IBM communications environment. As shown in Figure 1, there are four basic types of hardware components. These are remote terminal devices, which may consist of high-speed RJE stations; low-speed time sharing terminals, (CRT or hardcopy); remote CPU's; and a variety of other specialized remote devices (credit card readers, etc.).

These remote devices communicate with an IBM System/360 or System/370 host processor by means of either a 2700-series Transmission Control Unit (2701, 2702, or 2703) or a 3700-series Communications Control Unit (3704 and 3705). The 2700's are basic, hardwired controllers which contain a limited amount of logic peculiar to the characteristics of each line and/or terminal device. This logic includes the ability to recognize special control characters, the assembly and disassembly of characters, and certain line monitoring to time-out inactive devices. The 3700-series Communication Control Units are actually communication processors, but are programmed to operate as perfect emulators of existing 2700's by means of an Emulator Program (EP).

Connecting the host processor with a Transmission Control Unit is the multiplexor channel over which multiplexed messages (either in one or four-byte blocks) are transmitted back and forth to the network. The multiplexor channel, its architecture and command sequences are difficult to understand, and for many years have blocked development of effective non-IBM communication controllers.

In the host CPU, communications are currently handled by three basic pieces of software. First, there is the operating system (OS, DOS, or VS) which is linked to the user's application program via an access method (Basic Telecommunications Access Method-BTAM, Queued Telecommunications Access Method-QTAM, or Telecommunications Access Method-TCAM). The interface between the operating system and the access method contains linkages and protocols which, if violated, will "clobber" the operating system. The linkage between the access method and the application program is on a macro or assembly language level (with the exception of TCAM which incorporates a COBOL-level interface) and requires considerable skill by an application programmer to communicate with the network.

To transmit (or receive) a message, the application program must first frame the message with control characters appropriate to the destination device. Next, an access method macro subroutine is called to translate the message from System/360 or System/370 EBCDIC code to the code used by that device (i.e., a Device Dependent Module). The application program next begins to build the actual message in a buffer area; once the buffer is filled, a channel program starts transmission of the message to the network. In the meantime, the access method enters a "WAIT" state until transmission is complete before the next operation starts. The access method is also responsible for all basic teleprocessing functions such as routing, polling, error recovery, and device dependencies. Needless to say, all of these activities consume CPU cycles and core within the host processor.

TYPES OF CHANNELS

It is to this environment that the manufacturer of a non-IBM communications processor must address himself.

The input/output architecture of the IBM System/360 and System/370 supports three types of channels, each with its own operating characteristics. The three channel types are the byte multiplexor, selector, and block multiplexor (S/370 only).

The byte-multiplexor channel can operate in either of two modes, byte-interleave mode or burst mode. Operation in byte-interleave mode gives the appearance of multiple, simultaneously active devices on the channel to the programs executing in the host processor. This is accomplished by splitting all I/O operations into short intervals of time, during which only a segment of the available information is actually transferred across the channel interface. In fact, at any one instant, only one device at a time is logically active on the channel. The appearance of a multiplicity of simultaneously active devices is achieved by servicing

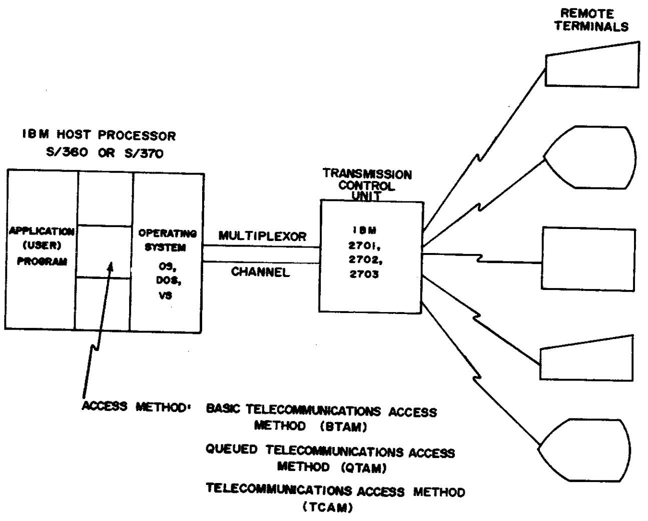

REMOTE TERMINALS
IBM HOST PROCESSOR S/360 OR S/370
APPLICATION (USER) PROGRAM
OPERATING SYSTEM OS, DOS, VS
MULTIPLEXOR
CHANNEL
TRANSMISSION CONTROL UNIT
IBM 2701, 2702, 2703
ACCESS METHOD: BASIC TELECOMMUNICATIONS ACCESS METHOD (BTAM)
QUEUED TELECOMMUNICATIONS ACCESS METHOD (QTAM)
TELECOMMUNICATIONS ACCESS METHOD (TCAM)

Figure 1—Existing IBM telecommunications environment

requests from a number of different devices at a rapid rate and by constraining each request to be very short in length (and thereby, in time).

Traditional IBM teleprocessing control units request service for one to four byte segments at a time. Obviously, there is some signalling overhead on each service request to identify which device it is associated with. The time for this overhead is appreciable, when it it multiplied by the number of segments required to comprise a message. This overhead tends to make byte-interleaving attractive for short messages from slow devices (such as keyboard interactive terminals) but not as economical for fast terminals which typically communicate long messages (e.g., remote batch terminals). Operation in burst mode is defined as dedication of the channel control functions to one device for the duration of an entire information transfer.

The selector channel operates in burst mode only. The utility of burst mode operation is that signalling overhead is only needed once per complete message transfer, instead of once for each tiny message segment as in byte-interleaving. This is why "fast" devices (e.g., tapes and disks) are usually run in burst mode. Telecommunications devices, on the other hand, do not usually interface to an IBM host processor in burst mode because of their slow operation (relative to internal host CPU and channel speeds). For example, a buffered terminal which "sends" a 1200 byte message to a host CPU over a 9600 baud line (assume 8-bit characters so that 9600 bits per second and 1200 characters per second and no other delays, such as line turnaround, modem latency, etc., are taken into account) would require one second of channel service time. During that one second interval, only about one thousandth of the capacity of the channel would have been used. Furthermore, if burst mode had been used, the channel would have been "tied up" servicing that device for the full second. All other devices attached to the channel would have been "locked out" for the duration. However, if the system operation were reorganized so the message was buffered in a device attached to the channel (instead of at a terminal at the other end of a communication line), then the limitation of the line speed would not be a factor in conveying the message to the host processor via the channel. Thus, burst-mode operation would be more applicable to telecommunications traffic.

The block multiplexor channel operates only in burst mode, but it has the capability to interleave blocks of data from multiple devices. This is done by servicing a pending request from another device, when the device occupying the channel in burst mode transfer reaches an end-of-block condition. The result is that block multiplexor channel operation has all the advantages of the high transfer rate associated with burst mode operation, and the disadvantages of channel lockout are reduced.

INTERFACING OPTION

From a user's standpoint, there are three basic methods of interfacing communication processors to IBM System/360 and System/370 channels and mainframes:

- Emulators
- Intelligent Emulators
- True Front-Ends

As its name implies, an emulator is a plug-for-plug replacement of an existing IBM control unit. To date, the three basic types of emulator on the market have met with varying degrees of success. These include the 2700/3700 emulator (270X/370X), the tape drive emulator, and the disk drive emulator. Of these, the 270X/370X emulator is probably the most popular and has met with the highest degree of success. By its very design, it interfaces perfectly with existing IBM telecommunications access methods and program products, and thus requires no change whatsoever in the existing IBM mode of operation.

For example, if BTAM is the access method used with an IBM 2703 or 3705, it remains present and unchanged when running with a perfect 270X emulator.

The 270X emulators can be divided into two categories; hardwired controllers (such as the Memorex 1270) and software-coded minicomputers (such as the Interdata 270X or the Digital Equipment PDP-11). Although the hardwired controller offers substantial cost savings, certain features are available (such as automatic speed detection) which give it limited advantages over the standard IBM 2701, 2702 or 2703. However, the advantages end there. The user is still forced to rewrite his host processor access method macros and change terminal adapters to support new devices and/or different applications.

Software emulators offer the user some cost advantages, but their greatest benefit lies in flexibility. Effective communications oriented real-time operating systems and modular device drivers make it possible to support non-

IBM-compatible devices by changing the communications processor program only. Thus, a programmable base exists for future expansion to permit the minicomputer to take over most of the communication control functions from the host IBM CPU. Thus, it serves as a stepping stone to a true Network Control Program form of front-end.

A second class of software emulator appears as a sequential device on the IBM selector channel. For the most part, these devices emulate IBM 2803 tape controllers, for example, Action Computer System's communication controller.

Processors which emulate 2314 disk control units have been somewhat less popular (e.g., the large Collins "C" System). Although these devices represent an attractive interface to user programs at the GET/PUT level, (thus eliminating the complexities of BTAM), sequential devices are designed for long message bursts at high speed. If considerable communication traffic is present in a network, it is possible as noted above to lock out other devices from the selector channel such as standard tape or disk drives, thereby limiting the user's access to mass storage media. The sequential device emulator, while thus attractive in certain applications, cannot be classed as a really effective solution to network control.

One step above a standard 270X type emulator, but not really a true "front-end", is the so-called intelligent emulator. This device usually consists of a programmable communications processor which appears to the host IBM CPU as a 270X, but which offers a number of enhanced features through programming. For example, non-IBM-compatible devices can be supported by such special software and thus made to appear to the host program as an IBM equivalent.

A second example is the "fail soft" protection against cessation of host processor operation. Should the IBM mainframe cease to operate, a special program module would allow the emulator to broadcast messages to all users telling them to discontinue traffic because the central site is down. Through the communication processor console, variable text could be inserted in such messages to inform the users when operation will resume.

A third example might be automatic port selection. In this case, all users would dial a single number for the host computer. By a simple sign-on procedure, the user would identify his device; the intelligent emulator would connect him with that "port" of the IBM CPU through which he can access the application he wants to communicate with.

A fourth possibility might be a local message switching capability. Each incoming message would be examined for a destination address; those bound for the host processor would be passed directly to it, while those bound for other terminals would be routed accordingly.

In all of these example, the communications processor appears as a perfect 270X emulator, only the *functions* performed by the processor differ. Therefore, without change to existing IBM telecommunications environment, users may realize considerable network flexibility above and beyond the control units supplied from Poughkeepsie.

NETWORK CONTROL PROGRAM

Finally, there exists the "true front end" or what IBM is promoting as the Network Control Program or front end NCP. Fundamentally, a communications front end is intended to remove the functions of controlling the telecommunications network from the host processor. The reasons for doing this are threefold.

The first is to reduce the cost of supporting a communications network. Most IBM teleprocessing installations have a significant portion of host processor resources tied up in terminal-dependent processing. Each software package that controls terminals has its own access method. For example, HASP users must use RTAM. CICS uses BTAM and special purpose applications; ATS has its own EXCP level programming built right in for terminal control. Accordingly, any installation supporting two or three teleprocessing applications usually has two or three teleprocessing access methods occupying core and using CPU overhead to execute.

A second motivation to front end an IBM processor is the desire to overcome the somewhat arbitrary S/360 and S/370 architecture restrictions, (they are not arbitrary from IBM's point of view; they are a fundamental marketing approach). For example, each terminal (line) is dedicated to a specific device address on the channel and thereby limited to one access method when the operating system is generated. To date, a single terminal could only communicate with one access method and thereby usually only with one application. Furthermore, since the terminals specific programming is provided in the host computer, and IBM primarily distributes software to control IBM terminals, the user cannot take advantage of the many benefits offered by the new terminals from other vendors. The terminal industry is currently one of the most rapidly changing segments of the computer field. Considerable advances are being made. Greatly expanded capability is being offered on new terminals by incorporating mini and micro processors right into the terminals at ever decreasing prices. Many IBM users would like to use these devices sooner than IBM is willing to let them.

The third motivation to front end is to make the network more easily accessible to application programmers. This means removing terminal I/O from the "bit banging" level of device-dependent modules and assembly language coding. Access to and from terminals needs to be provided to programmers working in higher level languages. Additionally, this type of software support should isolate the user, as much as possible, from the unique features of each terminal device. This device independence will make it easier to substitute new, improved, and lower cost terminals into the network.

From the standpoint of interfacing a front-end to an IBM host, the user must examine his individual requirements and decide which of the above problems he wants to alleviate. Then he can start selecting one of the many alternatives available to him. Unfortunately, the current state-of-the-art of software technology does not furnish a general purpose solution to all of these problems. Thus, the user is forced to decide which specific area or areas he wishes to attack, and implement his solution accordingly.

Some software technology already exists to ease the third problem, that of easing access to the network from higher-level languages. These are the TP monitors that are on the market. IBM's CICS and PMI's INTERCOMM are examples of these. They can be effective if a user can implement all his applications under control of the monitor. Then he only has one access method in his machine, and terminals can select which application they wish to communicate with. Perhaps in the future IBM will bestow upon the industry an improved access method which offers some of these advantages. Don't hold your breath waiting for it would be our advice, based on IBM's past record with teleprocessing access methods (remember QTAM?).

IBM users supporting a network of polled terminals usually suffer the most from the polling overhead being incorporated in the host. These users would realize an improvement in their operations by an intelligent emulator which does all polling and error recovery. In most cases, the host processor application would avoid drastic revision if it could be regenerated to use what it "thinks" are either unpolled terminals or the autopoll feature. The necessary programming could be incorporated into the intelligent emulator to achieve compatibility with the existing access method in the host. Following this approach usually makes it easier to support non-IBM terminals, since vendors of such systems are usually willing to offer this level of non-standard support as part of their product line. Additionally, some vendors offer support to enable the user to undertake this kind of activity himself provided the user has the expertise to do minicomputer programming himself.

SOLUTIONS

Limited solutions to the problems of overcoming S/360 and S/370 architectural limitations and IBM marketing strategy exist at present. Intelligent emulators can contend for channel device addresses based on activity when there are more lines than addresses. This is an interim solution which works well until a user wants more than about 160 active lines. Then he needs an access method which will map many lines onto a few channel addresses. Rumors of such an access method from IBM exist today. However, these rumors exist for S/370 users only. IBM has decreed that all its future enhancements will be restricted to virtual operating systems which only run on the S/370. People getting useful work done on S/360s, which are very cost-effective performers these days, are currently limited to emulators, intelligent emulators, and TP monitors to improve their efficiency. The best hope now is that after OS stabilizes for a period of time, some vendor will be brave enough to bring out a replacement access method the way some people are currently supplying enhancements to DOS. This would still be a mixed blessing at best, since existing applications would probably require a significant conversion and rewrite to utilize such a front end access method.

SUMMARY

In summary, in selecting a non-IBM communications processor, the user must take into consideration a certain number of basic factors:

- Perturbations to his existing applications environment
- Conversion to new hardware and software
- The impact upon existing IBM program products
- The effect on his overall telecommunications network
- Emulation versus intelligent emulation versus front-ending as a means of meeting his needs
- Support and maintenance
- Cost and benefits

Only when all these questions have been answered, can the user select that communications processor whose interface will truly match his environment, and which will effectively meet his needs.

BIBLIOGRAPHY

1. *S/370 Principles of Operation*, IBM Corporation, Armonk, New York.
2. Pryke, J. T. M., "A Front-End Primer for IBM Users," *Datamation*, April 1973.

Implications of changes in the secondary school—Mathematics curriculum for the computer science and computer engineering curricula

by M. E. SLOAN

Michigan Technological University
Houghton, Michigan

INTRODUCTION

Many professors of computer science and computer engineering are relatively unaware of the changes in the secondary mathematics curriculum in the last decade. The changes in mathematics have major implications for the design of computer science and computer engineering curricula in the colleges and universities. The thesis of this paper is that changes can be made in computer science courses to accommodate the changes in mathematics prerequisites and will result in strengthening the undergraduate program and allowing more time for advanced computer science topics.

CHANGES IN SECONDARY SCHOOL MATHEMATICS

In the early 1960s, a number of groups began working on major revisions of the elementary and secondary mathematics curricula. The most prominent of these groups was the School Mathematics Study Group, funded by NSF through the 1960s and early 1970s to combine the work of research mathematicians and high school mathematics teachers in a complete restructuring of the precollege mathematics curriculum.[1] SMSG initially developed curricular guidelines for the introduction of an approach to mathematics that would more clearly emphasize the structure of mathematics rather than routine computation. Realizing that the most effective way to get their ideas into the schools was to write textbooks and to train teachers to use them, SMSG initially devoted much time to the writing of texts that could be used in the schools and to the conducting of inservice training institutes to introduce teachers to their views of teaching mathematics. Later they concentrated on testing the effectiveness of their program versus more traditional programs and allowed other textbook authors to adapt SMSG materials freely. So pervasive were the SMSG ideas that by the time the project ended in the summer of 1973, SMSG director E. G. Begle of Stanford University commented that there were no mathematics texts in the country that had not been influenced by SMSG.

The changes in the secondary mathematics curriculum effected by SMSG may not be familiar to computer science teachers who graduated from high school ten or more years ago although they may know vaguely that even grade school students now spend much time on sets and less time on computational drill. Ten or fifteen years ago, the mathematics program of most high schools offered the college-bound student two years of algebra, a year of plane geometry, and a year of solid geometry and trigonometry. Some of the more adventurous schools offered an introduction to analytic geometry and calculus. All these courses comprised much the same material and approach since the turn of the century.

The SMSG approach centered on displaying the structure of mathematics. Perhaps most important to computer science curricula is the early introduction to concepts of discrete mathematics that are central to digital computation. Students are ordinarily introduced to basic ideas of sets in kindergarten and learn concepts such as set intersection and cardinality in the lower elementary grades. They also gain an early introduction to number representation and the use of number bases other than base 10. By sixth or seventh grade, most students have learned "clock arithmetic," an introduction to modular arithmetic. Computer-related concepts commonly taught in secondary mathematics are shown in Table I.[2-5] Many of these topics were previously not taught before the sophomore or junior year in college. The greater emphasis on discrete mathematics offers opportunities to accelerate development of computer science courses at the same time that the relative neglect of traditional computational skills, e.g., in trigonometry, necessitates greater remedial emphasis on prerequisite skills for courses like surveying in other disciplines.

A more recent development in secondary mathematics is the introduction of computer-based courses into the secondary schools.[6,7] There is not yet a standard pattern in the development of computer-based courses in the secondary schools; much depends on the individual interests of the teachers introducing these courses although several universities are attempting to provide standard training for teachers in their localities.[8-11] Basically the computer-

TABLE I—Typical Computer-Related Topics Taught in Secondary School Mathematics Courses

Grade 7	Numeration systems besides the decimal system
	Sets, including subsets, intersection, and solution sets
	Clock arithmetic (modular arithmetic)
	Number theory, including prime number factors, multiples
Grade 8	Equations and inequalities
	Elementary probability and statistics
Grade 9	Expressions, statements, and conditions
	2×2 systems of equations
	Mean, standard deviation
	Linear programming
Grade 10	Relations and functions
	Coordinate systems and distance
Grade 11	Relations and functions
	Linear programming
	Systems of equations
	Matrices
	Trigonometric functions and their graphs
Grade 12	Topics in linear algebra
	Topics in the elementary functions
	Topics in probability and statistics

related courses offered in secondary schools can be classified into two different types: mathematics courses with an algorithmic or computer-assisted approach and computer science courses.

Courses of the first type use the computer as a key to the teaching of mathematics. Either the computer is used to assist instruction in much the same way that it might be used to teach any other subject, or it is used to give greater insight into the details of mathematics by using an algorithmic approach to give the students more insight into underlying principles. These courses either use no programming language at all—students converse with the computer by simply responding to tutorial programs written in a tutorial language—or use a simple programming language like BASIC.

True computer science courses are being taught more frequently in high schools in what some reporters have even described as a torrent or an explosion. The most frequent computer science course taught is an introduction to programming and computers, similar to the B1 course in ACM 68.[12] Beyong this basic course, variations exist depending only on the whim of the instructor. Many secondary schools offer seminars in mathematics at the senior year; in several of these, various aspects of computer science are discussed. Engineering-oriented courses in computer science are possible as an outgrowth of courses in electronics; in these courses, students learn elements of computer logic and computer organization. Vocational courses in data processing teaching elements of COBOL and business techniques have been suggested; the author does not know of any high school currently offering such a course. Even such advanced computer science topics as operating systems, that three years ago had not yet been codified into a text, are successfully being taught to high school students.[15] At present, this is a very rapidly developing field which the university computer science instructor should try to follow.

Computer science courses in the high schools have not yet made much impact at Michigan Tech. A survey made by the author this fall of 71 students enrolled in a sophomore electrical engineering circuits course and 22 students enrolled in a senior electrical engineering logic design course showed that only 13 (15 percent) in the sophomore course and 3 (14 percent) in the senior course had taken a computer-related course in high school. As Michigan Tech draws its students primarily from rural high schools that lack the resources of urban schools, these figures are assumed to be an underestimate of the percentage of students in technical courses in the country who have had computer-related courses in high school.

In addition to changes in the formal secondary mathematics curriculum, many high school students learn computer science topics from what is sometimes called the informal curriculum. The informal curriculum comprises activities such as mathematics and science clubs as well as popular science magazines frequently read by high school students and general exposure to the news media. It is through the informal curriculum that many high school students learn of computers. This influence is so pervasive that it is doubtful that many scientifically oriented high school students anywhere in the country today are unaware of such basic facts of computer operation as their binary nature and the need for precise specification of tasks.

IMPLICATIONS FOR CURRENT COMPUTER SCIENCE COURSES

The main changes for capitalizing on the improved background of high school graduates in computer science and in discrete mathematics naturally occur in the first college courses in these areas. The most obvious improvement is to exempt students who have had programming in high school (or in the armed services or in industry) from beginning programming courses. Fortunately this is usually easy to implement. Most large colleges and universities offer a variety of beginning programming courses with variable credit or no credit. With good counseling available, a student should be able to choose the proper programming course or be exempted from the usual first programming courses depending on his background.

Another opportunity for building on the background of most high school graduates occurs in the introductory computer courses, whether service courses for nonmajors or the first courses for computer science majors that are more than just programming. A perusal of texts available for such courses will show that many texts are written with the expectation that students know little of modern computers. The majority of such texts devote unnecessarily thorough coverage to the fact that computers use binary arithmetic; some texts spend more than a chapter on the details of binary addition and conversion to decimal numbers. Since it can now be expected that nearly all students entering such courses will have dealt with binary numbers for several years, the treatment can be revised to a matter of fact review.

To illustrate the knowledge of typical electrical engineering students of simple calculations in binary arithmetic and number conversion, I gave a short unannounced quiz on the first day of class to students entering my sections of sophomore circuits and senior logic design. The results are shown in Appendix A; answers were counted as correct only if they were wholly correct. On this basis, about 40 percent of the students in the sophomore class, only 20 percent of whom had taken any computer-related course other than programming, could solve simple problems in binary addition and multiplication. A smaller percentage could convert numbers from or to binary, but nearly all were familiar with the processes and needed only brief review. Students in the senior course, all but one of whom had passed a course dealing in part with binary arithmetic through one's and two's complements the previous year, were only slightly more successful on binary addition and multiplication. The seniors did significantly better on the two's complement problem and on number conversion. While neither group did as well as might be desired, the sophomores without college training on these topics had a command comparable to the seniors.

The course on which the greatest gains can be made by building on the students' improved secondary school mathematics is the introductory computer science course in discrete structures, B3 in the ACM 68 curriculum. Yet this course often repeats secondary mathematics because the instructors teaching the course and writing the texts are not all aware of the amount of discrete mathematics taught in the high schools. For example, two computer science educators commented last year that the ACM 68 description of the B3 course is inappropriate since it begins with "Review of set algebra including mappings and relations" while the prerequisites in the ACM curriculum do not cover these subjects.[14] While this is admittedly true, the ACM 68 curriculum does not exist in isolation. The review suggested in ACM 68 should be a review of the material the student has learned in high school. The course should not be taught as if the students had not seen any of the concepts before; yet this is too often the case. It is more desirable to note as one author of a text for the course has done that the first material of the course (in this case the first chapter of the book) is covered in its entirety by the mathematics texts in statewide adoption in the California schools.[14]

In the computer engineering curriculum, the first course in logic or computer organization is often a place where savings in time can be made by capitalizing on the students' background in logic. The short entrance quiz, previously mentioned and shown in Appendix A, also showed that about 70 percent of the sophomores and 80 percent of the seniors could correctly write truth tables for AND and OR although less than 20 percent of either group had formally encountered this material in college. It seems likely that students have learned this material just as part of their basic cultural background and that they probably know many other simple logic concepts. Hence it seems that a more sophisticated approach to logic design and computer organization will appeal to most students. Unfortunately many current texts spend as much as two chapters in labored descriptions of elementary logic functions.

In any course, sound pedagogy requires the determination of the student's prerequisite knowledge. While it is often desirable to review prerequisites at the beginning of a course, it is important that a review acknowledge the student's previous contact with the material and not be a naive introduction. A review can build on a student's increased sophistication and mathematical maturity to provide increased rigor in an approach similar to Bruner's spiral curriculum.

IMPLICATIONS FOR THE FUTURE

The changes in secondary mathematics in the last decade exemplify changes we can expect in the future. While the content of the basic mathematics subjects has probably stabilized, changes in the use of computers in both basic mathematics courses and in computer science courses are occurring very rapidly. Closer articulation is needed between the high schools and the universities to improve the transition in mathematics and computer science. One possibility for increasing articulation would be a national study group with joint membership from both communities and with specific responsibilities for determining the best allocation of subject matter for each level. Such a group does not seem likely in the near future. Another possibility is for each university computer science department to establish an advisory committee of teachers from the high schools and the community colleges to keep it abreast of new curriculum and to recommend changes in the computer science curriculum. A third possibility is for colleges and universities to offer short courses in the use of computers in teaching mathematics for high school teachers and to see that such short courses allow two-way dialogues. A fourth way is for professors who make recruiting trips to high schools and community colleges to spend some time on their visits looking over the mathematics program. Fifth, all computer science faculty can establish informal contacts with secondary school teachers or students.

The increased activity in computer-related mathematics in the secondary schools is just one instance of the pattern of growth of computer science and computer engineering. These disciplines began as practical activities in the field to solve immediate problems. As the need for people who could design computers and program them grew, the graduate schools began teaching courses at a descriptive level and began codifying the material. As a theoretical basis for computer science grew and as the relationship with already established areas such as numerical analysis became more evident, undergraduate programs in computer science increased The current stage of growth represents more codification of known material into a form that can be taught at the secondary level; this stage is characterized by a marked increase in the number of texts available to high school teachers who do not ordinarily have the time or background to teach courses without texts and in an increasing interest of some computer science professors to develop courses for high school teachers and students. At the same time, research is exploring the frontiers of computer science and discovering material that in another few years will be conceptualizing the material and establishing a theoretical base that allows the material to be taught at earlier levels. At any time, of course, some older material becomes obsolescent and drops completely out of the curriculum. However, the growth of computer science is so rapid that it is essential to maintain efficient transitions from kindergarten to terminal degree and continuing education so that each student can effectively learn as much as he needs for his future.

REFERENCES

1. Wooton, W., *SMSG: The Making of a Curriculum*, Yale University Press, New Haven, 1965.
2. Johnson, D. C., et al., *Computer Assisted Mathematics Program (CAMP)*, Scott, Foresman, Glenview, Illinois, 1969.
3. Shanks, M. E., et al., *Pre-calculus Mathematics*, Addison-Wesley, Menlo Park, California, 1972.
4. Dolciani, et al., *Modern School Mathematics*, Houghton Mifflin, Boston, Massachusetts, 1967.
5. Wiebe, A. J., *Foundations of Mathematics*, Holt, Rinehart & Winson, New York, 1962.
6. Stenberg, W., "Computing in the High School—Past, Present and Future—And Its Unreasonable Effectiveness in the Teaching of Mathematics," *AFIPS Conference Proceedings*, Vol. 40, SJCC 1972, pp. 1051-1058.
7. Koetke, W., "The Impact of Computing on the Teaching of Mathematics," *AFIPS Conference Proceedings*, Vol. 40, SJCC 1972, pp. 1043-1049.
8. Atchison, W. F., "Computer Science Preparation for Secondary School Teachers", *SIGCSE*, 5, 1, pp. 47-50, February 1973.
9. Jansson, L. C., "Teacher Training in Computer Education," *SIGCSE*, 5, 1, pp. 47-50, February 1973.
10. Fu, K-C. and B. Koo, "Computer Science Education for an Overlooked Group—High School Teachers," *SIGCSE*, 5, 1, pp. 51-52, February, 1973.
11. McGinley, P., "The Training of Teachers in the Use of Computers in the Classroom," *SIGCSE*, 5, 1, pp. 53-55, February 1973.
12. ACM Curriculum Committee on Computer Science, "Curriculum 68," *Comm. ACM*, 11, 3, pp. 151-168, March 1968.
13. Booth, T. L., personal communication.
14. Engel, G. L., and N. D. Jones, "Discrete Structures in the Undergraduate Computer Science Curriculum," *SIGSCE*, 5, 1, pp. 56-59, February 1973.
15. Stone, H. S., *Discrete Mathematical Structures and Their Applications*, Science Research Associates, Chicago, 1973.

APPENDIX A

Percentage of Correct Responses to Entrance Test Given to Students in a Sophomore EE Circuits Course and a Senior Logic Design Course

Problem	Circuits Class	Logic Class
1. Change 451 in base 8 to base 2.	11	86
2. Change 110101111000 in base 2 to base 8.	11	77
3. Change 110101111000 in base 2 to base 16.	11	36
4. Multiply 101 by 110 in base 2.	41	50
5. Add 1101 to 1011 in base 2.	35	77
6. Complete this truth table.		
X Y X OR Y 0 0 0 1 1 0 1 1	72	82
7. Complete this truth table.		
X Y X AND Y 0 0 0 1 1 0 1 1	66	77
8. Change −25 in base 10 to two's complement notation.	6	32
Biographical Information		
Percent taking computer - related course in high school	18	14
Percent taking computer science course other than programming in college	23	23
Median age	19	22
Number (N)	71	22

A cognitive model for structuring an introductory programming curriculum

by CHARLES B. KREITZBERG

City University of New York
New York, New York
and
Educational Testing Service
Princeton, New Jersey

and

LEN SWANSON

EDUCOM
Princeton, New Jersey

THE PROBLEM

Introduction

From its inception the electronic digital computer has been involved in education, although its role has been the subject of some debate.[1] Academically, the computer has been used as a device for conducting or augmenting instruction, as a calculating device adjunctive to courses in engineering and the sciences, and most recently as an object for study in its own right.[2] Increasing attention is now being focused on undergraduate training in computer use, partly as a result of the recommendation of the President's Science Advisory Committee[3] that computing education be provided to all college undergraduates. We can expect that the number of students who undertake incidental study of programming as a part of their undergraduate curricula will continue to increase rapidly.

While the demand for programming instruction is increasing, little research has been conducted on how that instruction might be improved. There have been some investigations of programming behavior,[4] but these have mainly been concerned with how experienced programmers behave rather than how programming skills are acquired or how learning these skills may be facilitated. Many instructors express an uneasy feeling that the objectives of the introductory programming course are often not met; that many students fail to develop programming competency.

It is, therefore, important that we look for the sources of learning difficulties so that the courses we teach may be more effective. Motivational and instructional pacing problems, resulting from the frequent occurrence of students of unequal aptitude and experience heterogeneously grouped in large classes, clearly are factors. However, theories founded on educational psychology seem to suggest that more fundamental problems, arising from the absence of certain requirements for effective learning, are at play. It is the objective of this paper to propose a theory explaining some of these more fundamental problems, and to suggest the need for research supporting a methodology for overcoming them.

Rote and meaningful learning

Educational psychologists classify learning along several dimensions. One of the most important of these dimensions is the *rote-meaningful* continuum. Learning which is rote is verbatim memorization. In contrast, meaningful learning is non-verbatim; that is, the learner is able to restate the material in his own terms. Material which is meaningfully learned may be reformulated and *used* by the learner. In order to learn material meaningfully, the learner must integrate the material with ideas which he already understands. Thus the background of the student must be adequate if the material is to be meaningfully learned.

Rote learning and meaningful learning are not absolutes. Learning may be more or less meaningful or rote, depending upon a number of factors. The precise degree of meaningfulness will vary with the individual student. Factors which lead to rote learning tendencies are: material which is verbatim by nature (such as most foreign language vocabulary), lack of background on the part of the student, poor presentation of the material, and anxiety on the part of the student. A complete discussion of the *rote-meaningful* dimension of learning may be found in Ausubel.[5]

Numerous studies have shown that meaningfully learned material is remembered far longer, recalled with less difficulty, and utilized more effectively than is material learned in a rote manner. However, in order for a student to learn a subject meaningfully each fact to be learned must be integrated with the student's previous knowledge. Each new idea must be either an elaboration, an addition, a contradiction, or an example of an idea already in the student's background.

Learning to program is not typical of other learning because it has decidedly rote and decidedly meaningful components. The syntactical elements of the language cannot easily be presented in a meaningful manner, or related substantively to other constructs in the beginning student's experience. For example, it is difficult to see how the fact that a FORTRAN statement must begin in (or beyond) column 7 can be presented in a meaningful manner. It is an arbitrary fact which must be memorized by the student. However, the assembly of statements into a program requires a skill quite beyond the recall of arbitrary facts. It is impossible to learn how to program in a "verbatim" manner because programming is fundamentally a process of selecting and structuring statements chosen from those available within the language. Even general forms must be adapted to the particular requirements of the problem before they can be used in a program. For this reason it is important that the learning of programming be meaningful rather than rote.

We have pointed out that there are several factors which may cause material which *could* be learned meaningfully to be rote learned by the student. One of these factors is the lack of adequate background for the material, or *insufficient context*. Another factor which may inhibit meaningful learning is *insufficient vertical transfer*—an inability on the part of the student to transfer his understanding of a concept to an application of that concept in a program. A third inhibitory factor in the meaningful acquisition of concepts is a state of anxiety on the part of the student, which in this context we have called *computer shock*.

It must be recognized that these factors are not absolute. Material is rarely learned either completely by rote or completely meaningfully. In general, the rote-meaningful dimension is a continuous one, and the inhibitory factors to meaningful learning may operate to a greater or lesser extent depending upon factors unique to the individual student or to the learning situation. However, to the extent that these factors are operative they will tend to make it more difficult for the student to learn programming. Therefore, a programming curriculum which minimizes the effects of these factors should be more successful in meeting its objectives than alternative approaches. Before recommending a curriculum which minimizes these factors we will elaborate on them.

Insufficient context

Insufficient context results from a lack of related ideas in the student's background which can serve as a base ("anchor") for the new programming concepts and techniques being learned. Context derived from internalized concepts and ideas can provide substantive links between existing cognitive structure and the new meanings being encountered. For the beginning student context must be derived from prior knowledge and experience; as the student progresses context can be constructed from the set of basic and unifying principles that underlie the programming discipline, and as a result comprehension of more subtle concepts is possible.[5]

The lack of relevant prior knowledge on the part of the beginning student makes learning and retention more difficult. The presence of a sufficiently inclusive context makes it possible for new ideas to be reconciled with existing ideas and integrated into the cognitive structure more rapidly and efficiently. Bransford and Johnson have shown dramatically that establishment of context prior to exposing students to material to be learned facilitates both comprehension and retention of the material.[6]

Beginning students usually have few relevant anchoring ideas that could support efficient learning of programming. The study of a programming language is in this sense quite different from the study of a second verbal language. Bernard argues that the latter "consists fundamentally in the acquisition of an additional set of symbols for old, familiar meanings."[7] In acquiring a second language the student has already mastered basic concepts and syntactic code and thus has the necessary relevant context as part of his prior knowledge. In programming, however, the student has no easy referents for such concepts as the interchange of two variables or the searching of a matrix. These concepts thus form basic meanings whose learning must precede the purely syntactical concerns and provide the anchorage for subsequent learning. This explains why a programmer proficient in a single language can usually learn a second programming language very rapidly. The "vocabulary" and syntax of a programming language is usually quite simple once a basic set of constructs is learned, and the student has an appropriate context into which he can integrate the new set of symbols.

The lack of adequate context for the beginning programmer stems from the fact that a computer is unlike anything which the student has previously encountered, nor is programming very much like any problem-solving behavior in which the student has previously engaged. Yet computers are designed as they are for very good reasons—to provide them with certain well-defined capabilities. Unfortunately, it seems that most students are not presented with the relevant anchoring ideas that would help them see the language requirements as a necessary concomitant of the computer's functions.

Unless sufficiently inclusive unifying context is provided, a beginning student without the relevant anchoring ideas must either learn in a rote manner or press into service less relevant ideas, relying on his own ability to reconcile the new ideas with existing ideas in his cognitive structure. In the latter case the new ideas being encountered will be less firmly anchored and more quickly forgotten. If the student must learn by rote, deferring integration until he has acquired more background, then reconciliation and synthesis of ideas becomes difficult and his ability to apply his new "knowledge" is correspondingly impaired.

Insufficient vertical transfer

The second hypothesized source of difficulty in learning programming is insufficient "vertical transfer." Transfer is the facilitating effect of previous learning on new learning.

Transfer is usually defined as "horizontal" or "vertical." Horizontal transfer occurs when a previously learned task, A, facilitates the learning of a new (and different) task, B. Vertical transfer occurs when rules or concepts learned at a "lower" cognitive level (such as *concept learning*) are applied at a "higher" cognitive level (such as *problem solving*).* In the case of programming, the objective of instruction is to help the student move from the level of knowledge (of syntax) to the level of application, which is generally regarded as a higher cognitive level. In another context, the ability to play a musical instrument represents a (vertical) transfer from the level of knowledge (of musical notation and instrument mechanics) to the level of skill or application. In this case the skill is partly motor and partly intellective.

Transfer from the intellective to the skill level is essential when applying knowledge of a programming language to a specific problem. In most cases the beginning student's knowledge of the language consists primarily of vocabulary and syntax. Even if the solution to a specific problem is given to him in the form of a verbal algorithm—indeed, even if the implementation itself is shown to him—he has difficulty accomplishing the transfer because the gap between knowledge of vocabulary and syntax and problem solution is too great.

Gagné has said that "the components which appear to make problem solving possible are the rules that have previously been learned. Problem solving may be viewed as a process by which the learner discovers a combination of previously learned rules that he can apply to achieve a solution for a novel problem situation." (Reference 8, page 214)

Thus a proficient programmer, working on a problem that is within his experience, is hypothesized to operate primarily at a synthetic level. That is, he has learned or discovered the techniques or "meta-rules" that might be applicable to his specific problem, and applies these techniques in a relatively straightforward manner. It is precisely these meta-rules which permit him to operate at the level of synthesis much of the time, using analytic skills only when new constructs or techniques are required.

The beginning student, however, must try to break his problem down into a much finer set of subtasks that are recognizable to him in terms of his knowledge of the basic vocabulary and syntactic rules of programming before he can begin to develop the larger structures. In effect, he is constructing new techniques for solving non-intuitive problems, and this is an intellectually difficult task. Moreover, he must hold all potentially relevant rules in mind at one time when searching for those which might be applied to his problem.

The specific rules may not be obvious to the student because there are fundamental differences between human problem-solving techniques and computer solutions. People tend to solve problems by a "top down" approach, seeking patterns which may then be considered as problems of reduced complexity. Computers, on the other hand, have almost no ability to consider patterns, but rather operate in a sequential manner. As an example of this, consider the problem of determining which of the following numbers is largest:

1 12596 14

A person scanning this list would automatically and effortlessly pick out the second number as the largest, not because of its value but because of its length. A computer program to perform this same task would need to be written as a complex series of sequential operations; the following algorithm is an example:

```
i:=1; j:=a[1];
for k:=2 step 1 until n do
begin
    if a[k]>j then
    begin i:=k; j:=a[k]
    end;
end;
```

Such algorithms, while they are "natural" to the experienced programmer, are highly non-intuitive to the beginner who does not recognize the relevant rules. To expect the student to be able to create such algorithms without careful preparation is unreasonable; yet attempts are often made to teach this algorithmic approach by showing the student a complete program or two and expecting him to somehow absorb these non-intuitive concepts.

Even when a student is shown the solution to a problem he has difficulty transferring that solution to other situations.[8] A stimulus-response chain may be created, in which the student learns to respond in certain ways to certain problems without understanding the underlying rationale for the response. When asked to generalize from the solution, or solve related but perhaps more complex problems, he is at a loss.

The problem of learning from a given solution, or constructing generalizations of it, is particularly acute when the solution is complex or counter-intuitive, as for example in a *sort* routine. A study by Newsted suggests that there is a critical point of program complexity beyond which the student is unable to grasp the solution as a unit and requires additional documentary aids.[11]

It therefore seems reasonable to conclude that the learning of a programming language can be significantly improved if the student is encouraged to operate primarily at the level of rule application. To do this he must draw on knowledge of larger (non-intuitive) constructs than those represented simply by the vocabulary and syntax of the language. He must be able to subordinate his problem under such generalized techniques or meta-rules as sorting, interchanging, searching, etc. In a sense he must be presented with an extended "syntax" which includes these basic techniques or processes.

Computer shock

Beginning programming students are particularly susceptible to a phenomenon similar to one which in mathe-

* A number of authors have constructed cognitive hierarchies—learning tasks of increasing complexity. Two of the best known hierarchies are due to Bloom and to Gagné.[8,9] An analysis of programming texts in terms of Bloom's taxonomy may be found in Rademacher.[10]

matics courses is referred to as "number shock" or "number anxiety." The student tends to panic in the face of uncertainty about his understanding of the new and generally non-intuitive ideas he is encountering, and lacks confidence in his ability to apply these ideas to specific problems. The phenomenon is familiar to programming instructors who have seen students perform well on knowledge tests but struggle to construct even a relatively simple program. Faced with uncertainty and accompanying panic, the student tries to learn by rote lest he lose grasp of the material completely. The result is that synthesis of ideas is difficult if not impossible.

Computer shock may be partly due to general unfamiliarity with or lack of prior context for the subject. It is also caused by the necessity to operate with a degree of precision and at a level of detail which seems demanding to the student. We tend to forget how really technical programming can appear to the beginning student. He is exposed to a wealth of new information on complex machines and processes, and asked to operate in a cognitive mode which is different from his normal way of thinking. More important, he must learn a variety of rules and a difficult syntax before he has acquired the ability to apply his new knowledge in concrete situations. When he *is* asked to solve a specific problem it is usually of a complex variety which requires the synthesis of many ideas into a coherent solution. The result is an anxiety that makes meaningful learning difficult.

The three factors discussed above—insufficient context, insufficient vertical transfer, and computer shock—can make it difficult for the beginning student to learn programming. A number of factors, including the student's aptitude, the effectiveness of classroom lectures, and the availability of adequate laboratory facilities, will effect the ease and rapidity of learning. However, much can be done to overcome these specific bars to learning by improving the presentation of materials and methods of instruction. The following section suggests some guidelines for overcoming the problems discussed.

PROPOSED SOLUTIONS

Context

Since context is such an important part of learning, the organization of learning materials into discrete compartmentalized sections, each one containing all information relevant to a particular aspect of the language, will not improve learning and retention. On the contrary, in order to learn effectively the student must be presented with an idea and shown how it is similar to ideas he already has, how it differs, and how it extends, elaborates, or modifies his concepts. This makes it possible for the student to learn in a meaningful way and then use this new idea as an anchor for more elaborate ones.

The need for continually providing the student with adequate context suggests that a spiral approach, in which ever-increasing subsets of the language are developed, would be an optimal mode of presentation. A student will encounter an idea not once, but several times at increasing levels of complexity. Some attempts have been made to provide a subset[12] of the language initially and then progressively extend that subset; this is a useful approach but it is essential that the subsets be so selected that the student will see the logic of each extension. It is not sufficient to merely defer certain details to a later time. Each concept should be presented as an elaboration of the previously learned ideas.

Thus the student presented with materials in the "spiral curriculum" learns ever-increasing supersets of the subject as though he were walking up a spiral staircase, traversing the same points over and over again but at successively higher levels. The effectiveness of this approach has been substantiated by psychological studies.[5]

Learning materials should begin with concepts which are essential to an understanding of the subject. These concepts are of the following types: "What is a computer?" "What is a program?" "What sort of things can a computer do?" The introduction of these concepts must be carefully made, basing each new concept upon concepts with which the student is already familiar and showing the logical necessity of each new idea as a consequence of the properties that we desire in a computer. This minimizes the problem of learning in terms of contextual prerequisites.

In accordance with this philosophy, the introduction of new concepts should be preceded by *advance organizers* which provide a brief introduction to the concepts in terms of previously learned ideas. As the new ideas are then encountered the student has some notion of their relationship to other ideas with which he is familiar. A number of psychological studies (including the Bransford and Johnson[6] study cited earlier) have demonstrated that the use of advance organizers has a substantial effect on both comprehension and retention of the material that follows.[13]

Vertical transfer

The problem of facilitating vertical transfer, so that the student is able to utilize the concepts which he learns, can be resolved by judicious selection of exercises which implement fundamental programming structures. These exercises should drill the student in such areas as variable assignment, exchanging values, statement repetition, and basic program structure. The techniques which underlie the selection of these exercises can be found by inspection of professionally written programs to isolate common program structures utilized.

We have postulated the existence of "meta-rules" which are basic constructs which programmers use to build programs. An example of such a meta-rule in FORTRAN might be the zeroing of an array by use of a DO loop:

```
      DO 10 I=1,N
 10   A(I)=0.
```

or the interchange of two variables:

```
TEMP=A
A=B
B=TEMP
```

If such meta-rules can be identified and taught to the student he should be able to develop programs far more rapidly and with greater ease, since he no longer needs to invent these constructs each time he needs them.

Computer shock

The effects of computer shock may be minimized in several ways. First, the mystery which tends to surround the word "computer" should be dispelled as quickly as possible by describing the simple, readily understandable functions it is designed to perform and the manner in which these functions are built upon to create the more elaborate structures associated with actual applications. This will provide the student with a better sense of the capabilities and limitations of the computer, and place in perspective the reasons it is designed as it is.

Second, the basic operations the computer performs should be described in terms of similar operations with which the student is already familiar. The operation of an adding machine, for example, provides a good analogy for introducing the concepts of input, output, and program control.

Finally, written materials used in teaching should be as attractive and well-organized for instruction as possible. The materials typically found in reference manuals, for example, tend to be imposing to the student and are therefore unsuitable for instructional use.

SUMMARY

This paper has hypothesized three factors which inhibit the acquisition of programming skills. We expect that curricula which minimize the effects of these factors will be the most successful in teaching introductory programming.

Empirical evidence is required to support these hypotheses. Among the research which should be carried out are investigations into the specific concepts required to provide adequate context, identification of the meta-rules which will enhance vertical transfer, and construction of exercises which will facilitate the use and adaptation of the meta-rules in various contexts. Such research is much needed in a society in which "computer literacy" is rapidly becoming a requirement of the educated person.

ACKNOWLEDGMENT

The authors would like to thank Professor Ben Shneiderman for his contribution to some of the ideas formulated in this paper.

REFERENCES

1. Oettinger, A. and N. Zapol, "Will Information Technologies Help Learning?," *Teachers College Record*, Vol. 74, No. 1, September 1972, pp. 5-54.
2. Mosmann, C., "Computers and the Liberal Education," *The Educational Forum*, Vol. 36, No. 1, November 1971, pp. 85-91.
3. President's Science Advisory Committee, "Computers in Higher Education," U.S. Government Printing Office, February 1967.
4. Weinberg, G., *The Psychology of Computer Programming*, New York, 1971, Van Nostrand Reinhold.
5. Ausubel, D. P., *Educational Psychology: A Cognitive View*, New York, 1968, Holt, Rinehart and Winston.
6. Bransford, J. D. and M. K. Johnson, "Contextual Prerequisites for Understanding: Some Investigations of Comprehension and Recall," *Journal of Verbal Learning and Verbal Behavior*, Vol. 11, 1972, pp. 717-726.
7. Bernard, W., "Psychological Principles of Language Learning and the Bilingual Reading Method," *Modern Language Journal*, Vol. 35, 1951, pp. 87-96.
8. Gagné, R. M., *The Conditions of Learning*, New York, 1965, Holt, Rinehart and Winston.
9. Bloom, B. S., *et al.*, *Taxonomy of Educational Objectives: The Classification of Educational Goals*, New York, 1956, Longmans, Green and Co.
10. Rademacher, R. A., *Cognitive Levels in Computer Education for Business*, Ph.D. Dissertation, University of Nebraska, 1971.
11. Newsted, P. Unpublished manuscript.
12. Kreitzberg, C. and B. Shneiderman, *Fortran Programming: A Spiral Approach*, New York, in press, Harcourt, Brace, Jovanovich.
13. Ausubel, D. P., "The Use of Advance Organizers in the Learning and Retention of Verbal Material," *Journal of Educational Psychology*, Vol. 51, 1960, pp. 267-272.

On the preparation of computer science professionals in academic institutions

by J. A. ARCHIBALD, JR. and M. KATZPER*

State University of New York
Plattsburgh, New York

INTRODUCTION

One of the major problems facing the computing industry concerns the training of professionals. Many of the existing programs in computer science at academic institutions simply do not provide the preparation required for the practice of computer science in industry.[1,2] This problem is due primarily, but not exclusively, to the failure of our academic institutions to consider the needs of industry in ordering their priorities and goals. Other causes are the specific failure of industry to make its needs known in an adequate manner to the academic institutions, and the general lack of communication between these communities. Regretfully, neither community is taking adequate steps to solve the problem. In this void of activity, we find the several professional societies trying to bridge the gap with programs for the certification of professionals.[3]

This paper discusses the nature and implications of the situation, calls for a reordering of priorities within undergraduate programs at academic institutions, presents an applications oriented undergraduate program in computer science as a means of solving some parts of the problem, and makes specific suggestions of a number of other steps that may be taken in the industrial and academic communities to strengthen the profession at large.

THE PRESENT SITUATION

The disparity in the nature of computer science, as practiced in industrial and academic institutions, is abundantly clear to anyone who has participated in both areas. A major portion of the difference is the age-old dichotomy between the abstract and the applied. The primary requirement in industry is for applied computer scientists—people who can interact effectively with practitioners of other professions to develop meaningful solutions to existing problems within the limitations of present day facilities. The primary products of an academic institution are the theoreticians, capable of conceptualizing solutions to formally posed problems and relating these solutions to computational processes. Said in other words, industry needs people with applied backgrounds and academia is producing people with theoretical backgrounds.[2] Thus, the priorities are different. The industrial priority cannot be criticized, since it has passed the one universal industrial requirement—profitability. This profitability is the cornerstone of a thriving, new profession. The academic priority is also not wrong—it is the essential ingredient to the future of the discipline. The point is, however, that academia needs to recognize not only the priorities for the future, but also those for the present. It must increase its emphasis upon applications without sacrificing its emphasis upon theory. This can be done by tailoring the undergraduate program toward applications, and retaining the theoretical emphasis in the graduate programs. Undergraduate curricula should be geared to the majority that do not proceed on to graduate school rather than to the minority that do.

The fact that this dichotomy between theory and applications exists is not, of itself, bad. Indeed, there are many disciplines in which the divergence is greater than it is in computer science. The problem is one of timing. Our discipline was formed out of necessity. It is thriving because it was born with a "golden spoon in its mouth"—a large backlog of present day problems capable of solution within existing theory on existing facilities and literally crying out for solution now, in the present. The solution of these problems should enjoy a higher priority in our profession (particularly on academic campuses) than it presently has. Said in other words, our discipline is too young, and the pressures upon it to solve existing problems are too great for us to be able to afford the luxury of the current divergences between theory and applications. Divergence within the practice of a discipline is healthy and desirable once the discipline itself has matured. Ours has not, and will not mature until a sizable dent has been made in the backlog of problems.

The industrial world has only recently awakened to the potential of a computerized age—an age in which machines can and are both performing routine tasks of drudgery, and

* Presently with Ocean Data Systems, Inc., Rockville, Maryland.

providing formerly incomprehensible solutions to complex, practical problems. An age in which, however, computers are doing less than they should. The great challenge of industry is to harness the power of the computer in the service of mankind. Within industry, there is a need for people (primarily at the Bachelor's level) with the capability of joining and contributing to the problem solving team through the infusion of new ideas, techniques, and capabilities. This means people who have some understanding of both the problems of the industry concerned, and the methods of using currently available computers to solve these problems. This, in turn, means people who have combined a strong background in the area concerned with an ability to actually analyze problems, design and implement programs, and perform the other tasks related to the effective use of computers. These individuals must be versed in the various areas of computer techniques and analysis: e.g., statistics, simulation modeling, numerical methods, non-numerical methods, artificial intelligence and the like. They must understand such things as the effective organization of data, the effective utilization of processor time and memory, and the nature of multiprogramming computer systems. They must have a degree of both competence and sophistication in programming—the primary skill of computer science. These must be gained through in-depth training (in both classroom and laboratory situations) at the level of the current state of the art. In short, they must be competent applied computer scientists.

Our academic institutions have also, recently, awakened to a new age. They, too, have become impressed by the great potential of the computer. Their reaction to the challenge has been to plunge headlong into the development of advanced theories and courses which teach these theories, without concern for the practical uses of either the existing or the developing theories. Indeed, by virtue of their background, many have been too concerned with the new theories, and too disinterested in the practical uses of today's theories. The ACM Curriculum 68,[4] upon which many of our academic programs are based, contains numerous courses which use the current state of the art merely as a point of departure to enter upon abstract studies. These courses contain little emphasis or practice in the use of present technology, with the limitations of our present facilities. Thus, many of the young people who leave our academic institutions possess an outlook and orientation which are unsuited for doing the practical work expected of them in industry.[6] Industry has reacted to this situation by hiring college graduates in their major area of interest. These people are then hastily trained to do some programming. The results are, naturally, less than professional.

Another shortcoming of present academic programs is the tendency to concentrate computer science programs exclusively upon the computer, as though the computer were an end in itself. Hence, our graduates have little idea as to what the theories they have studied are good for. R. W. Hamming has referred to these non-professionals as "idiot savants" and "computniks."[5] Hiring such a graduate in computer science has become a burden that few of our industries with problems to be solved can afford.

REASSESSMENT OF PRIORITIES

The intent of this paper is to encourage the de-emphasis of theoretical computer science at the undergraduate level. Theory is important, for without it, the discipline will die, and many of the problems of the future will go unsolved. The plea is for a reassessment of priorities. Much more emphasis needs to be placed on applications in the undergraduate curriculum, with the emphasis remaining theoretical in the graduate curriculum. This re-directed emphasis on applications in the undergraduate program should have two thrusts: (1) the development of methodologies to be used in solving general classes of applied problems, and (2) the development of a level of competence in one or more of the disciplines to which computer science is applied.[5] We do, indeed, note some movement in this direction.[7] Our position is that this movement is insufficient. Under the present system, many of those who complete the undergraduate program are really unqualified for anything except graduate school. They go on to complete their graduate degrees with an esoteric thesis and little exposure to applications. Their training leads them to seek positions as undergraduate college instructors—they are unprepared for industry. As instructors on the undergraduate campus, they participate in the training of the next generation of undergraduates—again without exposure to applied computer science. And so, the vicious circle is closed. It continues to roll on by the sheer force of its own momentum.

This is not to take away from the importance of developing new theories. Indeed, such development is the vitality of a thriving discipline. Retention of theoretical emphasis in the graduate curricula should meet these needs. In other disciplines, where new problems are being solved in a timely manner as they occur, and where there is no backlog, an academic establishment concerned solely with speculative theories is ideal. However, in computer science, where there is a backlog, it seems appropriate to ask that the academic world devote a significant part of its effort to the present, the practical, and the applied. In short, there is a great need for the academic institutions to divert a greater part of their efforts to applied computer science. Theory can find its place in such a program in support of applications, in deepening understanding and in broadening horizons for imaginative solutions to problems. At the present, however, much theory is presented for its own sake. Significant applied and theoretical problems are being discarded for vacuous speculation. The result is that our academic institutions are not producing the kind of talent required by our industrial institutions.

The question as to whether academic institutions should produce people to meet specific demands of society, or merely educate people to enrich their souls, is an age-old question which will merely be mentioned here. We do live in a world of practicality, a world in which one must eat, a world in which the individual who can serve society has a higher survival potential than the individual who cannot.

THE VIEW FROM INDUSTRY

Industry, of course, is not to be denied. For one thing its position is vindicated in our society simply because its survival is based solely upon its ability to make profits. Industry will pay to get what it wants, or what it thinks it needs, in terms of manpower. It trains its own people when the skills it wants are not available. The problem here is that industrial training, by being oriented around corporate profits is generally inadequate, and usually falls far below professional standards.

Industry certainly needs people with, (1) an understanding of industrial problems, (2) an understanding of computers, and (3) fresh new ideas. Regretfully, the academic institutions are not producing sufficient numbers with this combination. As a result, industry hires graduates with majors in their areas of interest, and hastily trains them in computer skills. These graduates arrive at industry with, at most, two of the three major qualifications. The third one is developed, on the job, in a hit or miss manner. Industrial training does, indeed, tend to be very shallow. It is designed to meet certain very narrow objectives. Persons trained in industry are usually trained for a very specific task—with neither breadth nor depth. Such people must be continually retrained whenever their skills are required in a different activity. Industrial training is not geared to the production of professionals. These new employees certainly learn to program, but they tend to be weak in the areas of non-numeric methods, statistics, and the like. When an unusual problem arises they are not able to cope with it. This programming staff is often supplemented by graduates from two-year schools, who are often programmers with little understanding of more than straightforward coding, and graduates from the various training institutes. The training institutes themselves, however, are for the most part, designed to train technicians. The result is a total industrial staff that is less effective than it should be. It is distinctly subprofessional.

AN ACADEMIC RESPONSE

There is much that an academic institution can do to meet the needs of society. The new Bachelor's program in Computer Science at Plattsburgh State University College, included as an addendum to this paper, is one approach to this problem. Under this program, our computer science majors are required to develop a solid ability to do effective programming in at least two languages (one of which must be FORTRAN), to study the various areas of application and methodology of problem solving such as systems analysis, simulation, statistical methods, numerical methods, and non-numerical methods, obtain a modest theoretical background, and take an area of concentration in a discipline which uses the computer.[5]

The intent of this paper is not to present the program at Plattsburgh, but rather to encourage other institutions to devise their own approaches to applied computer science at the undergraduate level.

Specifics are indeed important at this point. Some of the features of the Plattsburgh program which, in the opinion of the designers are both important and unusual are listed below. We do consider these items to be essential in the proper training of computer science professionals.

Training in programming skills

We note an unfortunate disdain for courses whose primary objective is the teaching of programming skills in specific higher level languages. As stated above, programming is the fundamental skill of the computer scientist. We do not anticipate that our graduates will spend the major part of their time in actually writing instructions for a machine. We do feel, however, that skill and ability in the area of programming are essential if an individual is to perform any task related to the use of machines to solve problems effectively.

The first course in the major sequence at Plattsburgh concentrates primarily on the development of programming skills in FORTRAN. We use this language, despite all of its actual and theoretical deficiencies, because it is the primary language of the industry, and because there seems to be little chance that this situation is about to change in the foreseeable future. Our students are given instruction in the language itself as well as general instruction in computer science and programming. They receive numerous programming assignments, and, in addition, are required to complete an independent project in some area of interest to themselves. This course has a reputation of being difficult. There are numerous suggestions that additional academic credit be assigned to this course. Our experience with this course makes us wonder how effective programming and the numerous other topics that many institutions offer in their first course can be adequately covered in a single course. We use a tough two-semester sequence to offer, in greater depth, and with more actual programming on the computer, that which many other institutions offer in one semester.

We also have a similar course in COBOL programming. Most of our students use COBOL as their second language. Students in this course are given general information in computer science and programming, and specific instruction in COBOL, with emphasis upon the latter. Numerous programming assignments are also given in this course.

Area of concentration outside of computer science

The fact remains that computing is not an end in itself. Man does not compute for the sheer sake of computation, but rather because he is interested in something else—something in which computation is used to further understanding. Our students are expected to select some outside discipline to which computers may be applied, and to gain a depth of understanding that, while less than that of a major in the discipline, will be sufficient to permit the graduate of our program to interface with professionals of the other disciplines in the solution of problems of that discipline. Included in the possible areas of concentration are the natural, physical, biological, mathematical, social, behavioral, linguistic and administrative sciences. We have also included disciplines where the relationship is not quite as obvious, such as art and music. This is the feature of our program that, we feel, is a direct reply to Dr. Hamming's comments.[5] We note that certain other schools attempt to solve this problem by offering only graduate degrees in computer science.

Courses in techniques and methods

Our program includes courses in specific techniques and methods, such as statistical methods, numerical methods, non-numerical methods, systems analysis, and simulation and modeling, in which the emphasis is placed upon actual problem solving. Programming per se in FORTRAN is not stressed in many of these courses. We assume a background from previous courses sufficient for carrying out programming assignments. We stress the development of specific techniques, the use of specialized languages designed for the area concerned, and the effective use of library subroutines to solve specific types of problems of interest. Thus, the focus is upon the effective utilization of the computer, through whatever means are available, to solve problems of concern.

Courses in the fundamental nature and theory of computers

We do offer courses in machine and assembly language programming, programming languages, operating systems, and discrete structures. Certainly, no major sequence would be complete without courses in these areas. Our courses emphasize applications as much as possible. For example, in programming languages, we study the languages themselves, and their underlying structures from the perspective of how the language is used to solve the type of problem for which it was designed. Thus, for example, in the study of SNOBOL, we try first to understand the nature, applications, and problems of string processing, and then study the structures of the language as they relate to string processing. We do spend some time in a general discussion of abstract language structure, but we do not emphasize such considerations.

RECOMMENDATIONS FOR INDUSTRY

So far, we have focused our presentation on our sister academic institutions. Let us now consider the role of industry in meeting the problems of training the computer professional. If the academic institutions have been too unapplied, and too theoretical, certainly the reverse is true of industry. Too often the sights of industry have been narrowed to the area where the industry presently thinks that it can be profitable. These decisions of likely profitability are generally short range.

In general, industry is better able to carry out large scale advanced application developments. Industry has been deficient in both doing the application development work and in inspiring the academic institutions to do this work. Industry should interact more strongly with the academic world and, in doing so, get the academic institutions back on the track of reality. Specific suggestions (some of which have been discussed in the past[1]) include the following:

A. Providing of more grants to academic institutions to solve specific existing problems. Frequently, industry has the funds necessary, but neither the actual nor potential talent to solve its major problems. Academic institutions have few funds, but possess great potential for developing the necessary talent to solve these problems. The funds contained in a grant from industry would motivate the academic institutions to develop the needed talent. Academic institutions badly need the encouragement of industry to turn toward applications, and the availability of grants for specific purposes is the best way of providing this encouragement.

B. Providing computer time to academic institutions for use in developing techniques and methods for the solution of specific existing problems. Actually, this is another form of the grant suggestion made above. Relatively few academic institutions have the facilities needed to handle significant applied problems. This is particularly true in the area of scientific applications. Industry, by making its excess time available, and by indicating the kinds of problems that they are interested in having solved on the machine, can very effectively re-direct the academic institutions to significant applications.

C. Temporary employment of academic faculty members. One of the major causes of the current problem is the lack of communication between industrial problem solvers and academic faculty. There needs to be more contact between these groups. There should be a flow of both ideas and people between the two communities. Each community needs to know the problems of the other. Faculty members have always been available for temporary positions for durations of several months (one summer) up to fifteen months (two summers and the included academic year). The nature of college teaching lends itself to special summer work, and to sabbatical leaves. Very often, temporary employment for these periods is obtained at other academic institutions. Industrial institutions should avail themselves of the services of faculty members as consultants and in other special assignments of a temporary nature to aid this flow of ideas.

D. Making industrial personnel available to academic institutions either as temporary full-time faculty members or as part-time adjunct faculty members. This suggestion complements the previous one, and it has the same motivation, the flow of ideas and people between the academic and industrial communities. The temporary use of people on a full-time basis is the reverse of the academic sabbatical. It would be of great help in making the academic institutions cognizant of the problems of business, while enabling the academic institution to retain the flexibility of staffing that is so important in the modern college.

E. Providing "internships" with pay for students. This again may take one of two forms, the full-time employment of students in areas of interest for specified periods of time, and the part-time employment of students in areas of interest for longer periods. Certainly the best way to learn applications is to be involved in applications. This experience can be very meaningful in directing students to promising areas of application, and would be an ideal supplement to the instruction in methodology and techniques. Additionally, it would provide the student with badly needed funds. The actual employment, if over the summer, would be analogous to the conventional summer job. Full-time employment for a single semester would be analogous to the semester of student teaching that education students go through, or the internship served by new medical doctors. It would have the same benefits. Where conditions permit, half-time employment, and half-time course loads is another very viable possibility.

CONCLUSIONS

We, collectively, as practitioners of computer science, at both industrial and academic institutions, have a serious problem in terms of effective utilization of human resources. We must all dedicate ourselves toward working together to solve it if we are to make the contributions to civilization that we have been challenged with.

ADDENDUM—The Applications Oriented Undergraduate Program in Computer Science at Plattsburgh State University College

General Implementation

Students in the computer science major program at Plattsburgh are required to take a broad spectrum of applied computer science courses which include techniques and methodologies, a kernel of pure computer science courses which include concepts and theories, and a sequence of courses in an area of concentration, or a discipline with present or potential computer application. In our consultations with our students we point out the interactions between computer science and other disciplines. As a result of our applications oriented philosophy, our students are encouraged to devise study plans that combine investigations in computer and information science with the study of other fields. Simultaneously, they learn to apply the ideas of systems analysis to areas of contemporary concern. We emphasize the sequence in a related discipline as being the specific means of avoiding the lack of breadth which was the basis of much of the historical opposition to undergraduate programs in computer science.[1,5] Many of the courses included in our program are designed for both computer science majors and for students needing courses to support work in other disciplines.

The faculty at Plattsburgh is unusual in the large degree of interdisciplinary collaboration that goes on in research and teaching. This factor has aided greatly in the implementation of our computer science program. Our colleagues have joined us in advising students as to how to apply the knowledge obtained in computer science to yield interesting results in other disciplines. Concurrently, we have joined our colleagues in aiding them to include the use of computers in their own courses, and in their research. Many of the applications advocated at computer science education conferences and in the literature have been independently implemented at Plattsburgh.[8,9,10]

Specifics of Implementation

The emphasis in our computer science curriculum is upon applied computer science, i.e., upon the use of computer science to support work in other disciplines. Within the scope of applied computer science are both numerical and non-numerical applications. Numerical applications include numerical methods, statistical analyses of data, simulation and modeling, and applications to business and the quantitative sciences. The non-numerical applications include list processing, string manipulation, text editing, information storage and retrieval, and applications to the graphical sciences. Applied computer science is also involved in the mastery of techniques for organizing complex informational structures and the development of skills in analytic approaches to problems. Such skills are especially developed in the study of systems analysis and artificial intelligence.

There are a number of courses which we denote as pure or theoretical computer science courses. In order for any one discipline to be used effectively in support of other disciplines, its own development must have a certain degree of sophistication and maturity. Thus, the practitioners of computer science must have a certain competence in pure or unapplied computer science in order to apply it to advance other disciplines. Pure computer science includes such things as programming in machine and higher level languages, the study of languages and compilers, operating systems, discrete structures, computer logic, computer electronics and theories of computability and automata. We expect our majors to acquire a degree of competence in these areas.

In Table I, we present a general overview of the courses in our program. We note that the exact sequence for each student is determined on an individual basis depending upon the interests and capabilities of the student concerned as well as the area of concentration selected by the student.

Descriptive Listing of Generally Recommended Courses

Introduction to Electronic Data Processing: Introduction to computers, information science, data processing, concepts of hardware and software, cybernetics, and programming in BASIC language. Major emphasis on concepts of information science rather than on computer programming.

TABLE I

Usual Year of Study in the Computer Science Program	Courses
Freshman	Introduction to Electronic Data Processing (with BASIC) Systems and Information
Freshman or Sophomore	Introduction to Computer Science I & II (with FORTRAN) Second Higher Level Language Computers and Society
Sophomore or Junior	Machine and Assembly Language Programming Discrete Structures and Computational Analysis Numeric Methods Non-Numeric Methods Analysis of Statistical Data
Junior or Senior	Programming Languages Operating Systems Graphical Methods Artificial Intelligence Simulation and Modeling Undergraduate Independent Study Undergraduate Research

Systems and Information: The course will show how to study complex systems in terms of their interacting components. A general approach to such studies will be presented. This approach will involve analysis, modeling, and simulation of systems, together with information flow, control, and information processing within systems. Interdisciplinary skills are used for analysis and modeling in the social sciences, natural sciences and environmental studies. Contemporary examples will be taken from areas of concern such as ecology and economics. The role of information flow and control in systems is investigated. The processing of information by computer is demonstrated. An individual term project requiring the student to analyze a system of his choice is included.

Introduction to Computer Science I (with FORTRAN): Introductory course in computer science. Introduction to computers and use of computers to solve problems. Emphasis on introduction to FORTRAN programming. Numerous programming assignments. Development and implementation of independent project of the student's choice is required.

Introduction to Computer Science II: Intermediate problem analysis and FORTRAN programming, introduction to error analysis, numerical applications, non-numerical applications, operating systems, and machine and assembly language programming, survey of higher level languages and debugging techniques.

Second Higher Level Language: Introduction to computers and the use of computers to solve problems in the language concerned. Emphasis on programming in language concerned.

Machine and Assembly Language Programming: Elements and techniques of machine and assembly language programming as applied to hypothetical and actual computers, with emphasis on relationship to programming in higher level languages.

Non-numeric Information Processing: Fundamentals of non-numeric information processing, and programming techniques used in the solution of non-numeric problems. List and string processing languages and compiling techniques. Emphasis on applications.

Discrete Structures and Computational Analysis: An introduction to discrete mathematical structures with special emphasis on theories that are relevant to the study of computer science. Topics include aspects of logic, Boolean algebra, finite machines, switching circuits, set theory, induction, trees, strings, graphs and finite automata. Emphasis on applications.

Numerical Methods: An introduction to the basic notions of numerical solution of problems through an in-depth study of numerical processes. Emphasis is placed on errors inherent in computation and error analysis.

Computer Analysis of Statistical Data: An introduction to the use of a digital computer for the analysis of statistical data. The characteristics and significance of common statistical tests and distributions are studied through experimentation on a computer. Existing computer-library routines are used where possible. Application to problems in the student's major area is stressed.

Simulation and Modeling: Introduction to the use of computers for the simulation of systems studied in the fields of natural, social, and administrative sciences. Emphasis is on the formulation of problems and the design of models which reflect the activities of the systems concerned, and which lend themselves to being programmed for the computer. Probability functions, stochastic variables, Monte Carlo techniques, and queuing theory. Both continuous and discrete systems are studied in detail. Specialized simulation languages and statistical verification of simulation results are also considered.

Programming Languages: A study of the general principles and applications of computer languages and the techniques and problems of language implementation. Emphasis will be placed upon the use of significant languages excluded in other computer science courses, e.g., ALGOL, PL/I, APL, LISP, and SNOBOL.

Operating Systems and Systems Programming: Detailed study of the design and nature of the interface between man and computer: job scheduling, resource allocation, task overlapping, input/output handling, interrupt processing, multiprogramming, multiprocessing, time-sharing, assembly and compilation, and system subroutines.

Graphical Data Processing: Nature of graphical information processing and programming techniques used in the solution of these problems.

Artificial Intelligence: A survey of the application of heuristic programming techniques in computer game playing, problem solving, searching and pattern recognition. Principles involved in list processing languages will be reviewed. Machine learning approaches will be used for the writing of interactive educational programs.

Independent Study and Research: Project or study individually arranged between student and sponsoring faculty member. Results presented in seminars.

SUMMARY

We feel that the sum of the courses and requirements presented above yields more than the knowledge of a computer as a problem solving tool. Rather it imparts an ability to recognize and formulate problems, to seek solutions to these problems, and to make optimal use of the computer. This results in part from the emphasis that the faculty lends to the subject matter, and the interaction between students and faculty. A graduate with a bachelor's degree (in any field) will hold a subordinate position in his first job. As such, he will be carrying out tasks assigned by others. As a result of our program, our graduates will have an understanding of the requirements and interests of their superiors and will be able to carry out their duties in an appropriate and efficient manner.

REFERENCES

1. "Higher Education: Study Urges Altered Thrust in Federal Support," *Science*, Volume 182, Number 4112, November 1973.
2. "The Misdirection of Computer Education Efforts," *Journal of Data Education*, 1972.
3. Bylaws, Institute for Certification of Computer Professionals (ICCP).
4. Curriculum '68," *Communications of the ACM*, Volume 11, Number 3, March 1968.
5. Hamming, R. W., "One Man's View of Computer Science," *Journal of the ACM*, Volume 16, Number 1, January 1969.
6. Kandel, A., "*Computer Science—A Vicious Circle*," *Communications of the ACM*, Volume 15, Number 6, June 1972.
7. Austing, R. H. and G. L. Engel, "*A* Computer Science Course Program for Small Colleges," *Communications of the ACM*, Volume 16, Number 3, March 1973.
8. Blum, R., Ed., "Computers in Undergraduate Science Education," *Comm. on College Physics*, 1971.
9. Proceedings, *Conference on Computers in the Undergraduate Curriculum*, 1970-1-2-3.
10. Recommendations for an Undergraduate Program in Computational Mathematics, Committee on the Undergraduate Program in Computational Mathematics, May 1971.

EDP education—An acute crisis*

by GOPAL K. KAPUR

University of California
Livermore, California

INTRODUCTION

According to a recent AFIPS study,[1] the estimated number of unfilled openings for trained programmers in 1970 was between 200,000 and 650,000; and for systems analysts there were between 60,000 and 200,000 positions available. Industrial leaders are the first to admit that existing educational facilities are not adequate to supply the quantity and quality of training needed to fill the present market requirements—not to mention the steadily increasing demand foreseen for the future. From these observations it is obvious that the full potential of the computers presently in use cannot be realized because of the scarcity of sufficiently qualified programmers and analysts. As a consequence, many present-day computer installations are on the brink of chaos.

This shortage of sufficiently competent personnel hurts the computer users in more ways than just the imposition of artificial limitations on the utility of their computers. Since the demand exceeds the supply, job placement odds favor ill-qualified programmers and analysts. Such individuals change job after job, often landing better paying ones, and even more often compounding the miseries of their new-found employers.

The intent of the present paper is to examine the existing methods of training and to discuss the considerations involved in the development of educational programs for turning out highly competent and knowledgeable programmers, analysts, and data processors.

Without question, the most critical problems facing the people on the hiring (and firing) end of the EDP staff

- An inadequate supply of well-qualified programmers and analysts.
- An abundance of poorly trained, unqualified and inefficient personnel.

If the latter of the two observations seems unduly harsh, one need only consider the vast number of data processing installations that are approaching virtual collapse, caught months and years behind their schedules. A recent study by McKinsey & Company[2] of computers in 36 major firms states that "In terms of technical achievement, the computer revolution in U.S. business is outrunning expectations. In terms of economic payoff of new applications, it is rapidly losing momentum." It further states that ". . . from a profit standpoint, our findings indicate computer efforts in all but a few exceptional companies are in real, if often unacknowledged, trouble. Faster, costlier, more sophisticated hardware; larger and increasingly costly computer staffs; increasingly complex and ingenious applications: these are in evidence everywhere. Less and less in evidence, as these new applications proliferate, are profitable results. This is the familiar phenomenon of diminishing returns. . . ."

The March 25, 1970 issue of COMPUTERWORLD reported that McDonnel & Company, a 65-year-old securities dealer, will gradually liquidate its business. Reason: ". . . the failure of a new computer system to operate as it had planned, compounding the back office problems during the latter part of 1968 when stock brokerage volume was at a record level."[3]

Yet another example of mismanagement is contributed by the data processing department of the State of Massachusetts. According to a recent report filed by Donald R. Dwight, Commissioner of Administration, misuse and underuse have created a data processing wasteland, State legislators have charged, "mismanagement" of computer installations, and they plan to conduct public hearings on purchase procedures.[4] These are but a few of the many examples of doom brought on an organization by the data processors.

WHY SUCH A POOR SHOW?

Data processing, especially programming, has been and still is being taught as a trade. Most programming training is really an introduction to coding, and systems analysts are created by promoting senior programmers to new positions (rarely accompanied by any training in systems analysis). The educational institutions have also looked upon (or shall we say looked down upon) business data processing and analysis as technician's work, rather than an applied science. Little has been done to impart the discipline of a science to the data processing curriculum. The industry's record is no better. The recruiting, training, and treatment of EDP personnel have not been approached in the same manner, or with the degree of thoroughness, as is the case with engineers and scientific computing personnel. Very seldom does one

* Work performed under the auspices of the U. S. Atomic Energy Commission.

hear about bridges collapsing, computers shorting out, or buildings tumbling. Even the leaning tower of Pisa is taking its sweet time. But there are scores of writeups on EDP project failures each month in EDP news magazines and papers. The lack of proper education and training in the field of business data processing is one of the major causes for the sad status of the data processing industry.

The computer is one of the most precise machines ever developed. And yet, little effort has been spent in imparting disciplined education to the people who use this machine as a tool for information processing and management. Training has been haphazard, and it continues to be so even today.

There are four main sources of instruction for data processing personnel: private EDP schools, colleges and universities, manufacturers' schools, and in-house training within the industry.

PRIVATE EDP SCHOOLS

Over the past decade approximately 1,000 private EDP schools have mushroomed into existence throughout the country. This has become a $100 million-a-year business. Individual schools charge as much as $2,000 for a six- to eight-month course. Often the performance records of these schools leave much to be desired. Most of the EDP schools are franchised, and unfortunately the franchise is usually bought by a salesman or commercially oriented person who wants to make a fast buck and disappear from the picture. These schools give their students very little other than a framable piece of paper. Very seldom is the franchise owner a person dedicated to, or even interested in, the cause of decent education. The quality of education provided by most of these installations is substandard, and their overall training approach provides little preparation for the actual business environment for which the student is being prepared.

The past few years have seen an increasing number of instances where the malpractices of such schools have been brought to the attention of public authorities. During an interview the director of the Massachusetts Consumer Protection Division remarked, "I will close every computer school in the state, if necessary. . . " to stop misrepresentation and fraud.[5] Fraud and malpractice have also been brought to the attention of the authorities in Texas, California, and many other states. Certainly there are reputable EDP schools imparting good educations, but their number is small.

Private EDP schools try to teach too much too fast to students who are simply not qualified. Most of these schools cover FORTRAN, COBOL, RPG, and assembler language in a six- to eight-month period, and some even throw in a course in PL/1 for good measure. All this is directed at students who have passed a watered-down, so-called aptitude test. Personal interviews with students from different schools on the West Coast revealed that most had not received any classroom exposure to tape and disk file processing. The main emphasis is on theory, and very little is placed on the practical side of business data processing expertise. These students get precious little experience in structuring problems for solutions and almost no exposure to information processing applications.

The instructors in these schools are usually underpaid, they are seldom qualified, and only rarely are they competent teachers. Instruction usually concentrates on the mechanics of the given languages, with very little time or effort being devoted to the overall logic of the language, its application, and basic programming techniques.

CONTROL OVER PRIVATE EDP SCHOOLS

Private EDP schools are on the front line in the effort to meet the needs of the manpower market for entry-level personnel in the coming years. These schools merit careful and continuous evaluation and guidance. Until recently, industry as a whole has chosen to disregard these schools; thus their effect on the quality of the education offered has been minimal. The participation of industry is essential to the proper orientation, control, and effective monitoring of these institutions. Industry should press for tighter governmental controls over the licensing and operation of such schools. Through its professional organizations, industry should draw up workable and practical standards for private schools, and such standards should be made available to the public, the press, and—most important of all—to the vocational counselors.

Recent EDP school guidelines prepared independently by the ACM and DPMA represent a major step toward the standardization of educational benchmarks and goals. However, such organizations cannot by themselves legislate a workable solution, nor can they force individual schools into compliance. One obvious practical solution lies in industry's backing up its evaluations by providing legitimate employment opportunities to graduates of the schools that uphold high academic and professional standards. Furthermore, the data processing industry, as potential employers, should realistically evaluate their existing and future job requirements for trainee-level personnel. This should be conveyed clearly to reputable EDP schools.

WHAT ABOUT COLLEGES?

One of the greatest enigmas is the gross failure of colleges and universities to involve themselves in the development of business data processing science. The academicians are among the worst culprits in the unforgivable practice of imparting substandard and superficial instruction and representing it to be far more than it is. Only a handful of universities and four-year colleges are presently offering programs leading to degrees in business data processing, and very few have plans to implement such programs. Why have the pundits of higher education chosen to ignore such a vital field? Very few institutions offer courses in data processing systems design, data structures, information retrieval, analysis methods, programming techniques and applications, mathematics oriented to business data processing, and other related subjects. Usually the courses are aimed at engineering and scientific uses of the computer. The emphasis is on pure science, with only cursory attention paid to the development of applied computer techniques for commercial utilization.

One reason for the poor quality of courses and the inadequacy of instruction stems from the manner in which the faculty at such institutes gets its exposure to business data processing. One or two instructors (in the fields of management, law, accounting, marketing, etc.) leaf through this or that manual for a given language, write a few elementary programs, and—without further ado—a data processing professor emerges. They usually have only skeletal knowledge of the language and often no appreciation of the real world of data processing. Engineering, mathematics, or even basket-weaving departments would never allow so shoddy an arrangement. Nonetheless, these people are now acknowledged leaders in the academic world of data processing. Just because a person happens to have the title of assistant, associate, or full professor in law, marketing, management, or accounting, it does not mean that the same person, by reading a few manuals, has qualified himself as a professor of data processing. It is not uncommon to assign such teachers to data processing classes for the sole purpose of filling their minimum unit load requirements. These instructors are usually only about a semester ahead of their students. What a flagrant violation of a student's basic rights to decent education. This is not to suggest that they are incapable of teaching such courses, only that they should be required to get vigorous training and first-hand experience before taking up teaching assignments.

The educational community has failed to realize the importance of disciplined training in business data processing. Proof of this is the lack of adequate—much less excellent—courses in colleges and universities throughout the country.

The California State Universities and State Colleges have acquired a network of third-generation computers with a price tag of approximately $8,000,000. Nonetheless, only one of these schools, California Polytechnic State University at Pomona, provides a major in business data processing—and only at the undergraduate level. Of the other schools in the system, only California Polytechnic State University at San Luis Obispo offers a substantial number of courses in business data processing. The University of California, Kansas State University, Howard University, Georgia Institute of Technology, Ohio State University, and Brigham Young University are typical of a seemingly endless list of excellent schools that, while offering instruction in computer sciences, have not seen fit to offer many courses in business data processing.

In his keynote address to the ACM Conference on Personnel Research, Anthony Oettinger of Harvard stated, "We ought to help the programmer survive by proper education. But who can we look to for such education? Not the new departments of computer science in universities. These departments are just getting out from under the influence of competing engineering and mathematics departments, and they are too busy teaching Simon-pure courses in their struggle for academic recognition to pay serious time and attention to the applied work necessary to educate programmers and analysts for the real world." The degree of purity of the computer science that is taught," claimed Oettinger, "is inversely related to the competence of the department in meeting social needs." He further stated, "We don't need any courses in Latin to develop general powers of reasoning. . . ."[6]

INDUSTRY AND THE COLLEGES

In the long run, colleges and universities are the only hope for well-qualified business data processing professionals. The data processing community should exert pressure on the academicians for the rapid implementation of new courses and the upgrading of the existing ones. The business community can show its real interest by setting up scholarships and by contributing time, expertise, financial assistance, excess computer time, and whatever else might be available to help establish a business data processing curriculum and adequate facilities in colleges. The educators need this type of participation rather than donations of obsolete hardware that give the donors handsome tax writeoffs and the recipients perpetual headaches.

Colleges, on the other hand, should abandon their ivory towers as far as education in business data processing is concerned. They should seek advice and help from the industry in designing data processing courses. The academic community should not feel that asking for guidance in such matters from the people in the industry is belittling. The commercially oriented people are the ones who are faced with the problems. They know what basic knowledge and training is required. They know what general trends exist in the industry and what will be needed in the foreseeable future.

One reason for the shortcomings of business data processing courses in colleges and universities is the dearth of qualified faculty. To fill this void, serious consideration should be given to the possibility of employing knowledgeable people from the industry. These people may not fulfill the established degree and credential requirements, but they have much more to offer the student than a professor with a degree in accounting or law and a short course in a computer language. "A little knowledge is a dangerous thing."

The programs offered by colleges and universities should concentrate on business data processing systems design, analysis methods, programming logic and techniques, business mathematics, data structures and information retrieval, equipment evaluation and selection, and the design, evaluation, and implementation of software. Refresher courses for upgrading industrial staff members already working with the computer should not be neglected. Courses oriented to the problems of data processing management and executive education should be offered through seminars and workshops. If business data processing applications are to attain the same stature as other applied sciences, they will have to be accomplished in an environment conducive to such development, and they will require some formidable exertion on the part of educators.

Industry, on the other hand, should offer programs in which teachers from colleges can be given opportunities to obtain first-hand experience in practical data processing. This could be realized in the following ways:

- In-house training programs during summer or at other convenient periods to train the sort of data processing teachers the industry particularly requires. The businesses that are most influential in the future of EDP should take the initiative in this matter and establish lines of communication with various colleges and universities.
- Occasional workshops in the advanced "state-of-the-art" could be offered for teachers to keep them informed about the latest innovations and developments within the industry.
- Some larger commercial installations might see their way clear to lending various sorts of assistance to prepare students of selected colleges for the real world of computer applications.

The long-range benefits from such programs will outweigh by far the immediate expenditures incurred by the computer industry.

COMPUTER MANUFACTURER SCHOOLS

Objectively speaking, the training imparted by the computer manufacturers does not deserve many laurels, either. Nonetheless, the level and extent of such training has, in many ways, saved the business data processing industry from total disaster. Their efforts in developing qualified user personnel for their equipment have at least supplied the industry with a large number of coders and some programmers and analysts. However, the training provided by the manufacturer schools should be recognized for what it is: training in how to make *their* computer execute procedures. The emphasis is merely on teaching people the instruction set of a given language. Little or no emphasis is placed on basic programming logic, file structure, file organization, system analysis techniques, system design methods, efficiency considerations, and documentation methods. People so narrowly trained end up learning only the language instruction sets. They certainly are not taught to program well, since very little emphasis is put on the logic and structure of the computer language, even though logic is the backbone of good programming. Some of the computer manufacturers have been providing essentially a makeshift education in EDP—yielding quick, generally flimsy, results.

IN-HOUSE TRAINING: A FIRST SOLUTION

It is easy to call for upgrading of private EDP schools and the institution of new and better courses at colleges and universities. However, it would be a long time before such efforts would really start to pay off. For example, it takes many years before new courses can be taken from conceptual stages to full implementation. Then, there is the need for upgrading the programmers and analysts who, due to a lack of real education, are contributing to the delinquency of the data processing industry.

The internal turmoil, inefficiency, and overall confusion within the majority of data processing installations stems from the quality and extent of training given to in-house programmers and analysts. Top management has been led to believe that programmer training is a fairly straightforward and not-so-important matter. The usual sequence of steps for in-house selection and training of programmers[7] goes like this:

- Administering an aptitude test to a number of company employees.
- Selecting a group of people who happen to have done well on the above tests and who can be spared from their present jobs.
- Having them attend a two- to three-week class conducted by the computer manufacturer.
- Placing them in the data processing department to start programming.

This approach to programmer training is in many ways similar to preparing "instant potatoes". What you get is a substandard product; in fact, you have created the "instant programmer".[8] The entire approach is open to serious challenge.

Quickie training of this sort exposes the student to nothing more than how to code—not even how to code well. The main emphasis is on programming languages, and during the training phase an average student will write one or two elementary programs. Basic EDP concepts are completely overlooked, and the results are disastrous.

The first step in improving the performance of the data processing departments is to improve the in-house training methods. The categories listed below provide a frame of reference for identifying programmer training needs:

- Basic familiarity with computer hardware and general architectural concepts.
- Introduction to programming, the bases of systems analysis, operating systems, and software.
- Learning the structure, usage, logic, and application of the programming language.
- Introduction to good programming habits, documentation, program optimization, and program structuring methods.
- On-the-job training in programming application—starting with simple I/O oriented programs—to be reinforced with discussions on modular programming concepts, desk-checking, and debugging techniques.
- Periodic seminars, workshops, or discussion group meetings to explore such subjects as file organization and resources, decision table application, and other related subjects.
- Occasional discussions on the efficient usage of company software, various utility routines, and job control language.

It is realized that it may not be possible for many installations to institute a formal training program, and in other cases it may not be advantageous to go deeply into every suggested area of study. Reasons for this could be many: lack of funds, very small staff, lack of training material and personnel. However, most installations will need to develop some type of organized approach if they are to survive the increasing shortage of well-qualified data processors.

The question arises: How can an organization develop adequate, workable, in-house programmer training material? The points outlined below, with certain modifications and adjustments, may be the solution for many companies:

- Conducting formal classes in those areas that tend to be formalized and routine, e.g., programming languages, introductions to hardware and software. The scope of each class will depend mostly on the background and needs of the participants. The overall approach and organization of such classes should be flexible enough to accommodate participants with varying degrees of experience. Such classes could be supplemented by programmed instruction (PI) and audiovisual courses available from computer manufacturers and other EDP educational concerns.
- Holding informal one- or two-hour workshops once a month may be advisable for dealing with topics of limited relevance to the majority of data processing employees, e.g., sessions on program debugging, uses of certain utility routines, interpretation of dumps, etc.
- Company management should also explore the possibility of inviting experts from other companies, reputable computer consultants, and qualified teachers from local colleges to discuss such matters as data structures and information retrieval, file organization techniques, and design and handling of large computer projects.
- An excellent method to fire up individual interest is to set aside a one-hour meeting time, perhaps once every three months. Different subjects are selected by the participants, and each month a programmer prepares and conducts the meeting on one subject, followed by a question-and-answer period. It is surprising how much good material can be developed through such meetings and, at the same time it gives the participants a sense of accomplishment.

Similar training methods and approaches can be applied to the upgrading of the systems analysis staff. The following categories identify the training needs in general:

- Basic training in the use of existing hardware and software. It is very important that a systems analyst have a clear understanding of the capabilities as well as limitations of the company hardware and software. New systems analysts, especially the entry-level personnel, should be familiarized with basic data processing and computer concepts, computer programming, and the software in use.
- Training in analysis and system design. This phase of training may touch on such topics as feasibility studies, basic system design, system development, and system implementation fundamentals.
- The system staff should be kept cognizant of all software changes, i.e., changes in operating environment, new utilities, new procedures, etc.
- Professional development. This phase of training involves those areas of systems development that will help the analyst to develop better and more efficient systems. Specifics of this area of training could be: the use of decision tables, special system techniques applicable to problems at hand, training in various phases of company business, test data generation, and system testing techniques.
- Other short courses such as effective communication, effective writing, leadership, problem analysis, financial management and budgeting, and interpersonal effectiveness should also be looked into.

In-house training requires considerable thought, planning, and commitment by the top executives, time and effort by the data processing staff, and inevitably, a definite financial outlay. However, such efforts will pay big dividends: more efficient use of company hardware and software, better employee morale, and above all, a smoothly operating data processing department—to name just a few.

CONCLUSION

There is no one solution to the problems that exist, and will continue to exist, in the field of business data processing. However, if sincere and conscientious steps are taken by the industry and the educational institutes, the problems could be alleviated significantly. It is obvious that the industry stands to gain tremendously by working in partnership with colleges, universities, and private EDP schools for the purpose of establishing sorely needed guidelines and objectives.

The collapse of McDonnel & Co.,[3] chaos in the data processing department of the state of Massachusetts,[4] and the general confusion and mismanagement that abounds should serve as dire warning to industry of the great peril at its gates. Industry is limping along hoping that manufacturer's schools, colleges, universities, and private EDP schools will improve their education and training systems, and the education institutes are expecting the same of the industry. But problems will never be solved unless all parties come together and agree on a system of data processing education that is mutually beneficial to the educators, the potential students, and the industry as a whole.

REFERENCES

1. *AFIPS*, 1970.
2. *Unlocking the Computer's Profit Potential*, McKinsey and Company, Inc., 1969.
3. *Computerworld*, March 25, 1970, p. 25.
4. *Computerworld*, April 1, 1970.
5. *Computerworld*, April 15, 1970.
6. "ACM Conference on Personnel Research," *Datamation*, August 1968.
7. Kapur, G. K., "Sharpen your EDP Staff Through In-House Training," *Computer Decisions*, March 1971.
8. Kapur, G. K., "The Instant Programmer," *Datamation*, November 15, 1970.

An undergraduate/graduate program in information systems

by CHARLES J. TESTA

University of Maryland
College Park, Maryland

A recent report[1] by the Association for Computing Machinery (ACM) Curriculum Committee on Computer Education for Management (C^3EM) offered recommendations for graduate professional programs in information systems development. The integration of the disciplines of computer science and business administration led to the development of a curriculum whose objectives are: "(1) to develop a systems point of view, (2) to provide a conceptual basis for the analysis of information systems in large complex organizations, (3) to develop an understanding of how to create an economically viable and technologically feasible computer-based system, and (4) to provide experience in the implementation of a complex information system." Thus, the program is designed to prepare students for a career as a designer and manager of computer-based information systems. It is assumed that students entering this graduate program will have sufficient preparation in basic mathematics, operations research, statistics, psychology, economics, and computer programming.

In a subsequent report[2] the C^3EM proposed a curriculum for an undergraduate program based on the same general concept of the information systems specialty in organizations. In both reports the life cycle of an information system was viewed as consistng of an iterated process of information analysis, system design, and implementation. The graduate program was structured to include both information analysis and system design, but the undergraduate program offered two concentration options. The organizational option prepares students as computer users while the technological option prepares them for an entry level position as a programmer. Each option requires the student to choose a field of application in which he will complete 15 semester hours of course work or the equivalent of a double major. Since the two options will typically reside in the schools of business and engineering respectively, the undergraduate student is faced with "school requirements" as well as "university requirements." As a result, the undergraduate program imposes strict requirements on students and limits their flexibility.

It is the author's belief that an undergraduate program can prepare students for entry level positions in the field of information systems. Additional on the job training and/or advanced education should enable these individuals to make significant contributions to the development of effective information systems. If structured properly the undergraduate program can provide sufficient background to enable students to complete a master's program in one additional year.

BEHAVIORAL FACTORS

In the aforementioned ACM reports, the need for better understanding of human behavior in information systems was clearly expressed. For many years hardware/software problems demanded the most attention while "people" problems were often ignored. As a result, sophisticated information systems were often developed, but people experienced difficulty in interacting with these complex systems. Since information systems are used, operated, and maintained by people, it is essential that students develop an understanding of human behavior.

It is the author's opinion that the field of information systems must identify and apply the specific behavioral principles involved in system design. The design of effective information systems will only result if man's perceptual, cognitive, motor, and motivational capabilities are taken into consideration. For too many years, we have bemoaned the fact that a "communication gap" between the manager and the information system specialist prohibits effective system development. A better understanding of the manager's personality should help to narrow this gap.

Current societal problems cannot be packaged for solution by specific disciplines. Their solution depends upon the contributions of individuals from many disciplines who often speak different "languages." In an attempt to foster interaction across traditional disciplinary lines, the University of Maryland has recently undergone an academic reorganization. The new organization is based upon the grouping of related disciplines into Divisions in the hope that this will facilitate the joining of theoretical and empirical aspects of knowledge and enhance the ability to develop interdisciplinary programs. The departments of economics (ECON), geography (GEOG), government and politics (GOVT), information systems (INSY), psychology (PSYC), and sociology (SOCY), and the college of management (BSAD) constitute the Division of Behavioral and Social Sciences. This structure provides an ideal opportunity for students and faculty to cross traditional barriers to interaction.

TABLE I—Undergraduate Program

General Requirements for Bachelor's degree in Information Systems.
A minimum of 120 semester hours with 30 hours in general university requirements, 30 hours in electives, and 60 hours in courses constituting Groups I-IV below.

Group I	(Information Systems)	Credit Hours
INSY 200	Introduction to Electronic Data Processing Systems	3
INSY 201	Advanced Electronic Data Processing Systems	3
INSY 202	System Development	3
INSY 203	Systems Analysis	3
INSY 204	Management Information Systems	3
INSY 205	Computerized Planning Systems	3
		18
Group II	(Behavioral Sciences)	
BSAD 240	Personnel Management	3
BSAD 241	Management and Organization Theory	3
PSYC 100	Introduction to Psychology	3
PSYC 200	Personality and Adjustment	3
ECON 100	Principles of Economics I	3
ECON 101	Principles of Economics II	3
		18
Group III	(Mathematics and Computer Science)	
MATH 100	Calculus I	3
MATH 101	Calculus II	3
MATH 102	Calculus III	3
CMSC 100	Elementary Algorithmic Analysis	3
CMSC 101	Language and Structure of Computers	3
		15
Group IV	(Operations Research and Statistics)	
BSAD 110	Business Statistics	3
BSAD 210	Operations Research for Management	3
BSAD 211	Linear Statistical Models in Business	3
		9

Group V (Electives)—30 hours

At least 18 of the 30 hours must be taken at the 200 level or above.

Subgroup VA (Related Discipline)

Choose one group of three courses (9 credits) from the following disciplines:

1. Accounting and Finance
2. Operations Research and Statistics
3. Marketing and Transportation
4. Government
5. Sociology

See Table 2 for suggested courses in each of the related disciplines

Group VI (General University Requirements)—30 hours

At least 9 of the 30 hours must be taken at the 200 level or above.

Code:
Courses numbered from 100-199 are first and second year undergraduate courses. Courses numbered from 200-299 are third and fourth year undergraduate courses.

UNDERGRADUATE PROGRAM

As a result of the preceding discussion, it is proposed that undergraduate students choosing information systems as their major field of study be responsible for the body of knowledge contained in the groups of courses shown in Table I. Emphasis is placed on understanding the fundamentals of the courses studied in Groups II-IV and applying the acquired knowledge to the field of information systems. In recent years a trend has evolved toward a broad general education at the undergraduate level. In keeping with this philosophy, Group V (electives) and Group VI (general university requirements) allow the student flexibility in fulfilling the requirements for 120 total credit hours.

The following is a collection of descriptions of undergraduate courses in information systems. Each course description specifies prerequisites or preparation expected of students taking the course. In addition, references (textbooks) are included to assist in further identifying the subject matter of the course.

INSY 200. INTRODUCTION TO ELECTRONIC DATA PROCESSING SYSTEMS[3,4]

Description

The objective of this course is to develop an understanding of the digital computer and its role as a tool in the information systems of organizations. Topics include an orientation to the stored program computer and its use as an automatic data processing system, concepts and use of a problem oriented computer language (COBOL), and the impact of computer-based information systems upon organizations and their management.

INSY 201. ADVANCED ELECTRONIC DATA PROCESSING SYSTEMS[5,6]

Prerequisites: INSY 200 and BSAD 110

Description

Intensive study of advanced COBOL topics and computer applications in scientific, information, and control systems with particular emphasis on information systems. Hardware and software features of third generation computers. Topics include COBOL for tape and direct-access devices, management information systems, multiprogramming and multiprocessing systems.

INSY 202. SYSTEM DEVELOPMENT[7,8]

Prerequisites: INSY 201, MATH 102

Description

Techniques for identifying an organization's information requirements and methods to design systems to meet these needs. Emphasizes systems development process and participation of management in this process. Topics include systems methodology, data analysis, data processing files, systems technology, systems economics, systems management.

INSY 203. SYSTEMS ANALYSIS[9,10]

Prerequisites: INSY 202, BSAD 211, MATH 103

Description

Application of systems analysis techniques to the development of information systems. Measurement, simulation, and evaluation of information systems. Topics include the principles of systems analysis, simulation models, network analysis, hardware/software performance.

INSY 204. MANAGEMENT INFORMATION SYSTEMS[11,12]

Prerequisites: INSY 202

Description

Conceptual approaches for the analysis and design of management information systems (MIS). Methods used in designing, establishing and maintaining a data base for a MIS. The role of data management in information systems. Topics include the concept of MIS, information structures, elements of data management systems, query systems and report program generators.

TABLE II—Suggested Courses in Related Discipline at the Undergraduate Level

1. Accounting and Finance		
BSAD 100	Principles of Accounting	3
BSAD 101	Intermediate Accounting	3
BSAD 200	Cost Accounting	3
2. Operations Research and Statistics		
MATH 103	Linear Algebra	3
BSAD 212	Operations Research I	3
BSAD 213	Operations Research II	3
3. Marketing and Transportation		
BSAD 220	Marketing Principles and Organization	3
BSAD 221	Marketing Management	3
BSAD 230	Principles of Transportation	3
4. Government		
GOVT 100	American Government	3
GOVT 101	Introduction to Political Behavior	3
GOVT 200	Principles of Public Administration	3
5. Sociology		
SOCY 100	Introduction to Sociology	3
SOCY 101	Principles of Sociology	3
SOCY 102	Research Methods in Sociology	3

TABLE III—Master's Program

General Requirements for Master's degree in Information Systems. A minimum of 30 semester hours with 15 hours in the major field and 15 hours in a related discipline plus a comprehensive written examination taken at the end of all course work.

Group I	(Information Systems)	Credit Hours
INSY 400	Design of Large-Scale Information Systems	3
INSY 401	Information Systems Management	3
INSY 402	Application of Advanced Developments in Information Technology	3
INSY 403	Human Factors in Information Systems	3
INSY 404	Concepts of Information Systems	3
		15

Group II (Related Discipline)—15 hours

Choose one group of five courses from the following disciplines.

1. Accounting and Finance
2. Operations Research and Statistics
3. Marketing
4. Transportation
5. Management
6. Personnel
7. Economics
8. Psychology
9. Government
10. Sociology

See Table IV for suggested courses in each of the related disciplines.

Code:
Courses numbered from 200-299 are third and fourth year undergraduate courses. Courses numbered 400-499 are graduate courses.

INSY 205. COMPUTERIZED PLANNING SYSTEMS[13,14]

Prerequisites: INSY 202, BSAD 211

Description

An examination of the techniques used in the design and implementation of computer-based planning and analysis models. Formulation of mathematical models and analysis of information requirements for the data base to operate the model. Verification and validation of models. The use of management information systems to support corporate strategic and tactical planning. Topics include industrial dynamics, statistical analysis, forecasting techniques, and conceptual framework for planning models.

The undergraduate program, as constituted should provide students with the necessary prerequisites for related graduate study in management, personnel, economics, and psychology. However, if a student expects to choose accounting and finance, operations research and statistics, marketing, trans-

TABLE IV—Suggested Courses in Related Discipline at the Graduate Level

1. Accounting and Finance		
BSAD 201	Advanced Accounting	3
BSAD 202	Advanced Cost Accounting	3
BSAD 203	Financial Management	3
BSAD 400	Managerial Accounting	3
BSAD 401	Financial Administration	3
2. Operations Research and Statistics		
BSAD 410	Managerial Analysis	3
BSAD 411	Application of Management Sicence	3
BSAD 412	Management Simulation	3
BSAD 413	Management Science I	3
BSAD 414	Management Science II	3
3. Marketing		
BSAD 222	Marketing Research Methods	3
BSAD 223	Consumer Analysis	3
BSAD 224	Industrial Marketing	3
BSAD 420	Marketing Administration	3
BSAD 421	Advanced Marketing Research Methods	3
4. Transportation		
BSAD 231	Advanced Transportation Problems	3
BSAD 232	Urban Transport and Urban Development	3
BSAD 430	Management of Physical Distribution	3
BSAD 431	Transportation Strategies	3
BSAD 432	Product, Production, and Pricing Policy	3
5. Management		
BSAD 242	Organizational Behavior	3
BSAD 440	Behavioral Factors in Management	3
BSAD 441	Application of Behavioral Science to Business	3
BSAD 442	Management Planning and Control Systems	3
BSAD 443	Organizational Conflict and Change	3
6. Personnel		
BSAD 243	Personnel Management—Analysis and Problems	3
BSAD 444	Personnel Management—Manpower Procurement & Development	3
BSAD 445	Personnel Management—Manpower Compensation & Evaluation	3
BSAD 440	Behavioral Factors in Management	3
BSAD 441	Application of Behavioral Science to Business	3
7. Economics		
ECON 200	National Income Analysis	3
ECON 201	Intermediate Price Theory	3
ECON 202	Quantitative Methods in Economics	3
ECON 400	Quantitative Economics I	3
ECON 401	Quantitative Economics II	3

TABLE IV—Suggested Courses in Related Discipline at the Graduate Level (Continued)

8. Psychology		
PSYC 201	Personnel and Organizational Psychology	3
PSYC 202	Engineering Psychology and Training Models	3
PSYC 400	Quantitative Methods I	3
PSYC 401	Quantitative Methods II	3
PSYC 402	Seminar in Human Performance Theory	3
9. Government		
GOVT 201	Governmental Organization and Management	3
GOVT 202	Quantitative Political Analysis	3
GOVT 203	State and Local Administration	3
GOVT 400	Metropolitan Administration	3
GOVT 401	Scope and Method of Political Science	3
10. Sociology		
SOCY 200	Formal and Complex Organizations	3
SOCY 201	Industrial Sociology	3
SOCY 400	Intermediate Procedures of Data Analysis	3
SOCY 401	Practicum in Data Analysis in Field Research	3
SOCY 402	Computer Methods for Sociologists	3

portation, government, or sociology as his area of specialization it is recommended that nine of his 30 electives hours be utilized to complete the appropriate courses in Table II. In any event, careful planning and counseling is required at the undergraduate level to ensure adequate preparation for graduate study.

MASTER'S PROGRAM

The master's program in information systems is structured to provide students with increased skills in information systems and a specialization in a related discipline thereby preparing them for careers as designers or administrators of information systems. With the trend toward less rigidity in undergraduate education it seems more appropriate to encompass a related discipline at the graduate level. As such, the program requires students to complete the groups of courses shown in Table III. In addition, students are required to pass a final written examination which emphasizes the integration of course material.

The following is a collection of descriptions of graduate courses in information systems. Each course description specifies prerequisites or preparation expected of students taking the course. In addition, references (textbooks) are included to assist in further identifying the subject matter of the course.

INSY 400. DESIGN OF LARGE-SCALE INFORMATION SYSTEMS[15,16]

Prerequisites: INSY 204

Description

Application of systems analysis techniques to the design and implementation of large-scale information systems for organizations. Emphasizes systems concepts, user's requirements, and the measurement, coding, and classification of data. Programming techniques for large-scale information systems, including time sharing and real time.

INSY 401. INFORMATION SYSTEMS MANAGEMENT[17,18]

Prerequisites: INSY 205

Description

An intensive study of the functions, requirements, and problems of managers of information systems. Methods and models for evaluating information system performance. Data security, legal considerations, and social impact of an information system. Personnel requirements and documentation procedures.

INSY 402. APPLICATION OF ADVANCED DEVELOPMENTS IN INFORMATION TECHNOLOGY[19,20]

Prerequisites: INSY 204

Description

Equipment useful in implementing information systems including key-to-tape, key-to-disc, mini-computers and microfilm. Data communicating devices including telegraph, telephone, microwave, and broadband telephone. Applications in business information systems.

INSY 403. HUMAN FACTORS IN INFORMATION SYSTEMS[21,22]

Prerequisites: INSY 204

Description

Sensory, motor, and cognitive functions related to man's capacity to perform information system tasks. Man as an information processor and transmitter. Intensive study of relationships between man and the computer. Topics include measurement and psycho-physics, personnel subsystems, computer I/O devices, programming languages, time sharing and interactive programming MIS and management decision systems, mini-computer systems.

INSY 404. CONCEPTS OF INFORMATION SYSTEMS[23,24]

Prerequisites: INSY 203

Description

Thorough investigation of the systems approach to problem solving and decision making in large-scale systems. Emphasizes the interrelationship of the systems approach and the planning process. Techniques and methods involved in systems analysis. Topics include general systems theory, information theory, cybernetics, and decision theory.

Table IV offers suggested courses in the related disciplines at the graduate level. However, through counseling with a faculty advisor the student may choose any five coherent courses compatible with his career objectives.

CONCLUSION

The recommendations in this paper are consistent with the curriculum models proposed by the C^{3}EM. It has been a well accepted fact that the field of information systems requires the contributions of many disciplines. As a result, many undergraduate/graduate programs incorporate certain aspects of computer science, operations research, and other quantitative techniques within their purview. It is the author's opinion that the behavioral aspects of information systems design must form an integral part of such programs.

LIBRARY LIST

The following list is not exhaustive, but represents the author's attempt to compile a list of books which he deems valuable for the undergraduate and graduate courses in information systems described in this paper.

1. Anthony, R. W., *Planning and Control Systems: A Framework for Analysis*, Boston, Harvard Univ., 1965.
2. Ashby, W. R., *An Introduction to Cybernetics*, New York, Wiley, 1956.
3. Beer, S., *Decision and Control: The Meaning of Operational Research and Management Cybernetics*, New York, Wiley, 1966.
4. Berelson, B., and G. A. Steiner, *Human Behavior: An Inventory of Scientific Findings*, New York. Appleton, Century, Crofts, 1968.
5. Bisco, R. L., *Data Bases, Computers, and the Social Sciences*, New York, Wiley-Interscience, 1970.
6. Chapanis, A., *Man-Machine Engineering*, Belmont, Calif., Wadsworth, 1965.
7. CODASYL Systems Committee, *Feature Analysis of Generalized Data Base Management Systems*, New York, ACM technical report, 1971.
8. Davis, G. B., *Computer Data Processing*, New York, McGraw-Hill, 1969.
9. Dearden, J., F. W. McFarlan, and W. M. Zani, *Managing Computer-Based Information Systems*, Homewood Ill., Irwin, 1971.
10. Dickmann, R. A., *Personnel Implications for Business Data Processing*, New York, Wiley, 1971.
11. Forrester, J. W., *Industrial Dynamics*, Cambridge, Mass., MIT Press, 1961.

12. Gruenberger, F. (Ed.), *Critical Factors in Data Management*, Englewood Cliffs, N.J., Prentice-Hall, 1969.
13. Gruenberger, Fred, *Information Systems for Management*, Englewood Cliffs, N.J., Prentice-Hall, 1972.
14. Guide/Share, *Guide/Share Data Base Management System Requirements*, New York, Technical Report, 1970.
15. Hare, V., *Systems Analysis: A Diagnostic Approach*, New York, Harcourt, Brace, World, 1967.
16. Head, R. V., *A Guide to Packaged Systems*, New York, Wiley & Sons, 1971.
17. Head, R. V., *Manager's Guide to Management Information Systems*, Englewood Cliffs, N.J., Prentice-Hall, 1972.
18. Head, R. V., *Real Time Business Systems*, New York, Holt, Rinehart, & Winston, 1964.
19. Hillier, G., and G. Lieberman, *Introduction to Operations Research*, San Francisco, Holden-Day, 1967.
20. House, W. C. (Ed.), *The Impact of Information Technology on Management Operation*, New York, Auerbach, 1971.
21. Kanter J. B., *Management Guide to Computer System Selection and Use*, Englewood Cliffs, N.J., Prentice-Hall, 1970.
22. Kanter, J., *The Computer and the Executive*, Englewood Cliffs, N.J., Prentice-Hall, 1967.
23. Karplus, W. J. (Ed.), *On-Line Computing: Time Shared Man-Computer Systems*, New York, McGraw-Hill, 1967.
24. Katz, D., and R. L. Kahn, *The Social Psychology of Organizations*, New York, Wiley, 1966.
25. Kelly, J. F., *Computerized Management Information Systems*, New York, MacMillan, 1970.
26. Krauss, L. I., *Computer-Based Management Information Systems*, New York, Amer. Management Assoc., 1970.
27. Martin, James, *Design of Man-Computer Dialogues*, Englewood Cliffs, N.J., Prentice-Hall, 1973.
28. Martin, J., *Design of Real Time Computer Systems*, Englewood Cliffs, N.J., Prentice-Hall, 1967.
29. Martin, J. and R. D. Norman, *The Computerized Society*, Englewood Cliffs, N.J., Prentice-Hall, 1970.
30. Meadow, C. T., *Man-Machine Communication*, New York, Wiley, 1970.
31. Metzger, P. W., *Managing a Programming Project*, Englewood Cliffs, N.J., Prentice-Hall, 1973.
32. Miller, G. A., *The Psychology of Communication*, New York, Basic Books, 1967.
33. Murdick, R. G. & J. E. Ross, *Information Systems for Modern Management*, Englewood Cliffs, N.J., Prentice-Hall, 1971.
34. Optner, S., *Systems Analysis for Business Management*, Englewood Cliffs, N.J., Prentice-Hall, 1968.
35. Orlicky, J., *The Successful Computer System: Its Planning, Development, and Management in a Business Enterprise*, New York, McGraw-Hill, 1969.
36. Rosove, Perry E., *Developing a Computer-Based Information System*, New York, Wiley, 1967.
37. Sackman, H., *Computers, System Science, and Evolving Society: The Challenge of Man-Machine Digital Systems*, New York, Wiley, 1967.
38. Sackman, H., *Man-Computer Problem Solving*, New York, Auerbach, 1970.
39. Schoderbeck, P. P., *Management Systems*, New York, Wiley, 1971.
40. Shannon, C. E. and W. Weaver, *The Mathematical Theory of Communication*, Urbana, Illinois, The Univ. of Illinois Press, 1964.
41. Sharpe, W. F., *The Economics of Computers*, New York, Columbia Univ. Press, 1969.
42. Simon, H. A., *The Shape of Automation for Men and Management*, New York, Harper & Row, 1965.
43. Weinberg, Gerald, M., *The Psychology of Computer Programming*, Cincinnati, Van Nostrand Reinhold, 1972.
44. Withington, F. G., *The Organization of the Data Processing Function*. New York, Wiley, 1972.

REFERENCES

1. Ashenhurst, R. L. (Ed.), "Curriculum Recommendations for Graduate Professional Programs in Information Systems," *Comm. ACM*, 15, 5, May, 1972, pp. 365-368.
2. Couger, J. D. (Ed.), "Curriculum Recommendations for Undergraduate Programs in Information Systems," *Comm. ACM*, in press.
3. Sanders, D. H., *Computers in Business*, McGraw-Hill, New York, 1972.
4. McCracken, D. D. and U. Garbassi, *A Guide to COBOL Programming*, Wiley, New York, 1970.
5. Murach, M., *Standard COBOL*, SRA, Palo Alto, Calif., 1971.
6. Naftaly, S. M., B. G. Johnson, and M. C. Cohen, *COBOL Support Packages: Programming and Productivity Aids*, Wiley, New York, 1972.
7. Clifton, H. D., *Systems Analysis for Business Data Processing*, Wiley, New York, 1969.
8. Gildersleeve, T. R., *Design of Sequential File Systems*, Wiley, New York, 1971.
9. Couger, J. D., *Systems Analysis Techniques*, Wiley, New York, 1973.
10. Gordon, G., *System Simulation*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
11. Lefkovitz, D., *File Structures for On-Line Systems*, Spartan, New York, 1967.
12. Lyon, J. K., *An Introduction to Data Base Design*, Wiley, New York, 1971.
13. Morton, M. S. S., *Management Decision Systems*, Harvard University, Boston, 1971.
14. Wheelwright, S. C. and S. Makridakis, *Forecasting Methods for Management*, Wiley-Interscience, New York, 1973.
15. Blumenthal, S. C., *Management Information Systems: A Framework for Planning and Development*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
16. Yourdon, E., *Design of On-Line Computer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1972.
17. Canning, R. G. and R. L. Sisson, *The Management of Data Processing*, Wiley, New York, 1967.
18. Myers, C. A. (Ed.), *The Impact of Computers on Management*, MIT Press, Cambridge, Mass., 1967.
19. Martin, J., *Telecommunications and the Computer*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
20. Martin, J., *Future Developments in Telecommunications*, Prentice-Hall, Englewood Cliffs, N.J., 1971.
21. DeGreene, K. B. (Ed.), *Systems Psychology*, McGraw-Hill, New York, 1970.
22. Sackman, H. and R. Citrebaum, *Online Planning: Towards Creative Problem-Solving*, Prentice-Hall, Englewood Cliffs, N.J., 1972.
23. Buckely, W., *Modern Systems Research for the Behavioral Scientist: A Sourcebook*, Aldine, Chicago, 1968.
24. Johnson, R. A., F. E. Kast and J. E. Rosenzweig, *The Theory and Management of Systems*, McGraw-Hill, New York, 1973.

Understanding the software problem

by JOHN B. SLAUGHTER

Naval Electronics Laboratory Center
San Diego, California

INTRODUCTION

In addressing the question of the high cost of software it is essential, first of all, to surround the issue in such a way that it is possible to achieve some understanding of the problem and its causes. The first question to be answered is that of whether or not software costs are, in fact, too high. In order to answer this question requires a definition of software itself. Continuing this vein of questioning obviously can lead to a circuitous path which never leads to anything but tangential conclusions about pseudo problems and pseudo causes.

Many articles, pamphlets, and government documents have been written about "the software problem" and what can be done to solve it. The fact that there is so much attention attributed to this concern is, in fact, indicative that some form of problem is extant and that the future of computer utilization is strongly dependent upon the manner in which software developers and users direct their energies to solve it.

Computer technology has advanced to an era where hardware development costs are declining per unit of capability. Extrapolation tends to indicate that for the foreseeable future, at least, this trend will continue. Conversely, software development costs usually measured in terms of lines of validated code per man-hour expended, are rising and expected to continue to rise. It is enlightening to examine the reasons why this difference exists and attempt to discern what inherent properties the hardware development process has that are not similarly possessed by software production. There are three major characteristics that can be identified immediately. They are design procedures based upon engineering discipline, highly developed automated manufacturing processes and technology, and the existence and use of design and production standards. None of these exist in a macroscopic sense for software.

A successful design effort must begin with a carefully conceived requirement that can be stated in the form of performance and design specifications which can be understood and followed by design engineers. Furthermore, it allows the project manager to keep track of the development and maintain control of the activities. This does not exist in the realm of software development. The reasons are several. Managers generally do not understand software and are unable to generate adequate specifications for its performance and design. Neither are they very capable of managing its development since, in comparison with a hardware device, it is an intangible quantity whose final form defies description in an engineering sense.

The absence of agreement on languages, programming aids, documentation methods, etc., for software is another major problem. Software development is a highly individually oriented activity where the programmer's skills, biases, and motivation govern the process. Automated program development is used by a small minority of programmers; most coding is performed by pencil and program worksheet. The wide range of abilities found in the members of the programming profession, the absence of a prescribed curriculum of educational experiences which are required for a person to qualify as a programmer, and the relative immaturity of the software profession all combine to create this facet of the problem.

The absence of design and performance standards which are used to govern software design is a key reason for some of the software development difficulties encountered by industry and government. Such standards are required if modular programs are to be developed since interfaces must be defined and met and care must be taken to assure total system compatibility of the component modules. Standards are also the essential link in software transferability from one application to another. The cost of generating a new program module each time it is required to perform a function could be eliminated or at least reduced greatly if adequate design and documentation standards were applied.

It is obvious that much of the software problem is due to the non-existence of the discipline and rigor which characterizes hardware design and production. Steps are being taken to introduce these needed features into the software process, e.g., structured programming, and the results are expected to provide improvements in software quality in addition to reductions in software development costs.

During the workshop sessions it was stated by one participant that the overwhelming characteristic of software is uncertainty. What constitutes reasonableness in terms of performance, cost, and schedule? The unavailability of sound answers to questions such as these is a major concern to developers, purchasers, and users of software. An attempt to define approaches to obtaining answers was a major theme of Workshop No. 1.

THE SOFTWARE PROCESS

Rather than attempt to state or develop a dictionary definition of software, it seems more meaningful and expedient to define the elements of the software process. These elements can be viewed as line-item cost factors that might appear in a budget for a major software system development. Thus they provide a very useful means of identifying where costs occur in the process.

	PHASES		
Elements	Development	Maintenance	Major Redesign
Define			
Analyze			
Design			
Code			
Compile			
Integrate			
Test			
Document			
Manage			
Provide facilities			

THE SOFTWARE PROCESS

The production of software requires the successful completion of several specific activities. In the foregoing chart ten activities have been identified as cost elements which must be taken into consideration in the development of a software product. Included among these are the elements of software production management and the provision of facilities and equipment on which the development will be accomplished. These are two significant factors which must be taken into account in any analysis of cost even though they are usually not addressed in most examinations of the investment required to produce a software output.

The chart also delineates three phases which occur during the life-cycle of any sizable software product. They include the initial development of the software which has been fully designed, tested, and documented for some particular application. The cost associated with this function is only a fraction of the total cost which must be allocated to the product. Also the maintenance of the software, i.e., the correction and elimination of faults which are discovered only after the package is put into use, minor improvements which are made to enhance the operation of the program, and expansion required to allow the program to work with minor changes in function, procedures, or equipment, must be considered as an important component of the total cost algorithm. Finally, major redesign or revision of the software is often required to accommodate major changes in mission, equipment, or operating doctrine. The extent to which the software is designed and documented initially to allow the maintenance and redesign functions to be accomplished easily are major considerations which must be addressed. Life-cycle cost, therefore, is the sum of the individual costs associated with each of the three phases.

Each of the cost elements will appear to some extent in each of the three phases. The weighting of each element will no doubt differ depending upon the phase being considered. Although time was not available during the workshop to quantify the extent to which a particular element is involved in the three phases, it was the consensus that such an effort could be very productive in focussing on where major cost problems occur.

As stated in the introduction, uncertainty is one of the major deterrents to a clear definition of the software problem. This is due in part to the paucity of data which can be used to examine where costs are incurred in the software process. An approach such as that described above, although perhaps somewhat simplistic in the exact form shown, would be useful as a means of acquiring needed information and providing a basis for improvement.

SOFTWARE PROBLEMS

What are the problems associated with the development of software? How do these problems manifest themselves and what are their causes? In answering these questions it is useful to assume the perspective of the individual or agency that contracts for a software development. The problems that are viewed from this vantage point are basically the following:

- Software Quality
- Life-Cycle Cost
- Delivery Schedule

It is obvious that these three factors are not independent variables and that perturbation of any one of them has a significant impact on each of the others. Nevertheless it is possible to examine each of them separately and attempt to identify the form in which the problem appears and some of the prominent reasons for its occurrence. From such an analysis, it is possible that the shape of potential solutions may become evident and that future research and development can be directed toward solving the basic problem.

Problem: Software Quality

Symptoms:

1. Unreliable
2. Unresponsive
3. Incompatible
4. Non-adaptable
5. Non-transferable
6. Uncertified

•
•
•

Principal Causes:

1. Inadequate statement of requirements by user
2. Inadequate understanding of user requirements
3. Poor testing and certification practices
4. Lack of standards by which performance can be measured
5. Inadequate documentation
6. Lack of appropriate management attention and control
7. Improper use of current technology
8. Inadequate programmer skill levels
9. Inadequate hardware/software trade-offs
10. Lack of adequate support software

-
-
-

Problem: Life-Cycle Cost

Symptoms:

1. High initial development cost
2. High operational and maintenance costs
3. Poor utilization of machine resources
4. Costly to modify
5. High cost of documentation

-
-
-

Principal Causes:

1. Poor estimation of production costs
2. Poor procurement practices
3. Poor software development practices
4. Lack of automated programming techniques
5. Improper or non-use of existing developments
6. Inadequate system hardware
7. Inadequate programmer skill levels
8. Poor system requirements and specifications
9. Lack of management control of costs
10. High salaries of programmers
11. Uncertainty of cost allocation
12. Inadequate attention to system integration and testing
13. Poor documentation practices

-
-
-

Problem: Delivery Schedule

Symptoms:

1. Failure to meet schedule
2. Long development time
3. Untimely software documentation

-
-
-

Principal Causes:

1. Poor estimation practices
2. Inadequate definition/understanding of job
3. Variable programmer skills and productivity
4. Poor management control and monitoring
5. Unrealistic milestones
6. Inadequate use of existing developments
7. Long lead-time procurement
8. Inadequate support software
9. Lack of automated programming activities
10. Inadequate attention to documentation

-
-
-

The repetitious presence of such factors as insufficient requirements, inadequate attention to testing, documentation, and integration, poor software management, lack of support software, and utilization of outdated techniques and tools points to these as the primary woes of the software community.

Many solutions to these problems have been postulated but their success requires a significant change in thinking by the developers and users of software. For example, the concept of structured programming in which software teams of specialists are formed is perhaps an approach which strikes at the heart of most of the faults attributed to the software generators. Structured programming follows a top-down design approach which inherently possesses an effective managerial structure, an ever visible and current set of documentation, an emphasis on testing and integration, a clear delineation of responsibility, and the development and utilization of new tools essential to the completion of the project. Experience with structured programming teams has shown that productivity of an order of magnitude can be achieved by the use of such a team over that obtained from a well-qualified individual programmer.

Changing user's habits may be more difficult. A more knowledgeable and aware set of project managers must be developed to generate meaningful requirements and specifications for software, to understand the dynamics of the hardware/software trade-off equation, and to provide the overall direction necessary to successfully integrate a complex software program with state-of-the-art hardware.

PROPOSED RESEARCH EFFORTS

One of the key causes of the uncertainty characteristic of software development is the absence of meaningful standards. Not only are there inadequate measures of performance, there are also inconsistencies in the jargon of the profession. Productivity of programmers is difficult to define because of different yardsticks. A standard data base is needed to provide benchmarks which allow software people to communicate meaningfully and effectively.

The question of standard languages is a particularly meaningful one to address. When one considers the number of different languages available (FORTRAN, COBOL, JOVIAL, CMS-2, PL-1, . . .) and the many versions which exist for any one of these, an appreciation of the lack of language standardization is readily gained. There is a strong and direct relationship between the lack of language standards and the cost of software. It can be measured in terms of the price of additional documentation, non-transferability of code, compilers, etc., development of new and often unique tools, unreliability, and the resultant large software inventory required. There is a need for standard operating system interfaces which can achieve a reduction in errors over that obtainable with current job control languages. A standard data definition language which eliminates the need for many data structures is another important requirement. Research into the costs of language standardization and how languages can be brought closer together represent sensible efforts to be initiated.

Workshop No. 1 addressed several "causes" of the software problem to attempt to develop approaches which could be recommended as potential means of obtaining solutions. The three chosen for consideration were:

- Lack of uniform measures of performance
- Inadequate exchange of information within software community
- Non-transferability of technology developments

It has been stated earlier in this paper that software development and procurement are handicapped by the absence of a meaningful and uniform system of metrics. Because of this, it is not possible to quantitatively measure the performance of either the programmer or his product and be assured that the reported values can be understood by others in the software community. As a result of this situation, it is difficult to identify what, if any, progress is being made in the area of increasing programmer productivity, for example. The recommended action to overcome this situation is the initiation of an R&D program designed to establish measures of programmer productivity. The approach to be followed is to collect objective statistics on such items as errors/lines of code, lines/day/programmer, and program documentation. In order to collect accurate statistics and make the results of the research meaningful, automated aids which accumulate statistics and on-line code production facilities are needed.

In spite of the numerous journals, conferences, symposia, seminars, etc., which address various aspects of software technology, working-level programmers are not sufficiently aware of the contributions which currently available techniques and tools can provide. Neither are they able to articulate precisely the help which they feel they need. It seems reasonable to make a concerted effort to identify users (persons who write software) and ascertain the form of the problems which they encounter and the kind of tools which they perceive would be useful to them. The recommended action is that there be established a 4-6 person ad hoc group consisting of tri-service, ARPA, FCRC members who during a period of 6-8 weeks will do a nation-wide study on the following questions.

- Who creates software? Where?
- What software functions are performed (requirements, analysis, coding, documentation, . . .)?
- What are the applications (Scientific, MIS, C^2, . . .)?
- What tools are in use?
- What tools are perceived as needed?
- What problems or difficulties exist?

Strongly related to the previous two issues is the question of transferability of technology developments throughout the community. Because there is no efficient information network linking software developers, useful and available tools are not used widely, and feasible techniques and tools are often not carried to completion. To solve this problem it is recommended that a network-based (ARPANET, for example) software evaluation and exchange facility be established. This facility would perform the following roles:

- Evaluate new software tools
- Disseminate evaluation results and experiences of other users to requestors (a system of charges to use the services can be established)
- Provide for general information exchange on techniques, standards, languages, etc.

CONCLUSIONS

Near the termination of Workshop No. 1, the question of what future course of action, if any, should be followed was discussed. Unanimously it was felt that some follow-on program was required in order to further the coupling of the deliberations of our meetings to users. The suggestions ranged from scheduling a regular series of meetings of our workshop group to establishing a full-time DOD activity to address the important issues of software. Certainly the sponsors of the symposium will want to consider a wide range of possibilities. It is believed by the participants of our workshop that a valuable exchange of information and ideas occurred as a result of the symposium and we would like to thank the sponsors and the chairmen for providing the opportunity. I would like to thank each member of my workshop for his (and her) candor, enthusiasm, and support.

Automated monitoring of software quality*

by J. A. CLAPP and J. E. SULLIVAN

The Mitre Corporation
Bedford, Massachusetts

THE PROBLEM

Widespread acceptance of computer systems for commercial and military applications has led to widespread dissatisfaction with the software components of these systems. Major criticisms have focused on the high cost of development and maintenance, the slippages in delivery dates, and the poor quality of software. Those of us who produce software readily admit to these complaints since we ourselves have to rely on software to help us do our job and have experienced the same problems. We are equally ready to acknowledge that the solution to these problems lies in better management of software development, but we quickly claim that software development is harder to manage than other kinds of production processes.

We know from experience that it is notoriously difficult to determine, prior to a software development project, just what product and product quality to expect with given resources, or vice versa, what resources will be needed for a given product. Even while a project is in progress, we do not know how to determine just what kind of product is emerging, how far along it is, and therefore how to project the (perhaps all too visible) resource expenditures that have already occurred to the final cost and time, so that plans can be altered and adjustments made. The result is that it is easy to be overambitious, and not even realize it until it is too late to do anything but sacrifice quality or commit still more time and money, and hope for the best.

Not surprisingly, such cost and schedule overruns—in conjunction with the poor product that is the usual result of such a badly controlled process—come to be regarded by customers as exemplifying the "high cost of software." But the notion that the cost is high is really only a conjecture; all that is known for sure is that, for most large, complex software systems, we do not know how to estimate cost, the proper relation of cost to quality, or even what quality is, and are therefore frustrated in our attempts to manage.

The problem comes down to a lack of generally applicable measures and measure-relating theorems that are useful to the manager in planning and controlling software development. Hoare,[1] in calling for a "software engineering discipline," and Boehm,[2] commenting on the factors of software costs, have said the same thing: we lack quantitative data and a systematic body of knowledge that will allow us to bring experience gained in one area to bear on a new situation in another area.

WHAT WE HOPE TO DO ABOUT IT

Our objective is to provide measures that give clear and current visibility to a software development project so that managers can know the real status of the software at all times and spot potential trouble in time to do something about it, and programmers can be made aware of the effects of their work. Making measurements implies the collection of data. It is often impractical or impossible to collect the necessary data manually because it is too costly or the data lacks the consistency and reliability which is required. However, the production of software has a distinct advantage over other kinds of production: much of the actual work takes place within a computer, and thus status information as well as data about the characteristics of the emerging product can be easily captured automatically and analyzed.

For these reasons, we decided to design and implement an automated software implementation monitor, called Simon, which extracts measures continuously throughout a software development project, stores data in a centralized data base, and provides analyses of the data to managers and programmers. In short, Simon's direct purpose is to provide project visibility in a practical, that is to say automatic way. In the long run, over many projects, the data gathered by Simon should become a part of our experience base, to support research into software quality and cost, and the causes thereof. Simon's contribution is to automate, control, and standardize the process of data collection and analysis. Equipped with observations and measures, we can relate pre- and post-implementation costs and final product quality to controllable variables during development.

WHAT IS SIMON?

Overview

Simon is an automated aid that is integrated into a conventional programming environment in the sense that it runs under the Operating System and invokes existing tools

* This work is sponsored by the Rome Air Development Center under Air Force Contract F19628-73-C-0001.

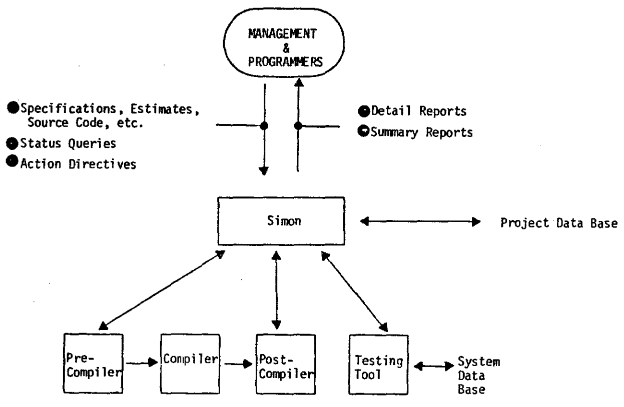


Figure 1—A configuration for the Simon system

such as a text editor, compiler, debugging tool, etc. In general, any transactions between the software builders and the computer are filtered through Simon to provide the following advantages: (1) Simon can monitor what is being done, by whom, and when; (2) standard procedures can be enforced; and (3) additional information can be requested of a user which would not be required by the operating system or other facility services. The data gathered before, during, and after a transaction is recorded in a data base which is structured to permit analyses to be performed and reports generated about any aspects of the data which are of interest.

Monitors are now in use which collect data about software automatically but these tend to serve specialized purposes, such as analyzing software performance characteristics, or hardware resource usage. In a few cases, monitors maintain a data base about the characteristics of the software under development. For example, the Automated Software Evaluation System[3] collects such data but limits its analyses to supporting software evaluation and validation. Still other software packages analyze and report on cost data to show managers where estimates and actual values deviate. All of these capabilities and others would also be incorporated into Simon in the belief that the whole can be greater than the sum of its parts.

The design of Simon is open-ended so that it can be adapted to different programming environments and extended as we gain experience in its use and as new techniques for software development emerge.

At the time of this writing, only a general design of Simon has been completed. A refinement of this design (in top-down fashion, naturally) will lead to the development of a version of Simon tailored to our particular programming environment and the measures of current interest to us. The description of Simon which follows is, of necessity, quite general. It is intended to convey the important concepts in its design and the kinds of capabilities it might offer, in order to provide a framework for future implementations by ourselves or others.

The specific method of implementing Simon will naturally be dependent on what data is desired and can be captured from existing sources, and what data must be generated by Simon. Figure 1 illustrates a typical Simon configuration of four modules: a pre-compiler, a post-compiler, a module testing tool and a monitor. The information flow among these units, the units native to the operating system, and the managers and programmers is also outlined. This configuration will be used as the basis for the descriptions which follow.

Functions

Simon has four principal functions, viz:

1. Edit
2. Compile (Static Module Analysis)
3. Test (Dynamic Module Analysis)
4. Status Report

Editing is the direct entry by project personnel of Simon file data, including documentation and certain status data such as estimates, schedules and costs, as well as source code. As a side-effect of editing, in the case of significant products such as documents and program modules, simple measures are extracted to project whether and when the change process will converge.

Static module analysis is the extraction of information derivable from the source (or object) code without actually running it or simulating a run. This is the kind of information that is available at compile time (or at edit time, but as a practical consideration many compilers already provide much of the information we want). Such information includes: file and variable set/used lists (concordances), inter-module reference graphs, the several measures of module complexity and data element complexity, independent path analysis (for testing), and static resource requirements (e.g., module memory size).

Dynamic module analysis is the extraction of information normally available only when the module is running—i.e., an instrumented test. This includes paths tested and success-failure data (for reliability estimation) and dynamic resource use (e.g., execution time).

Status reporting, the heart of the matter, is the systematic dissemination of summaries of the information collected, together with projections based on that information (including estimates and budgets). This implies that all pertinent information about a project is maintained in central files. There are in fact three basic kinds of files in Simon: (1) entity files, (2) relational files, and (3) event files.

Data base

Entity files track the current status of all entities thought to have a significant bearing on the project. An entity is any item that has duration in time, and properties that change over time without disturbing the basic unity of the entity. There are eight entity classes:

1. the project itself
2. people (programmers and managers)
3. program modules

4. documents
5. system tests
6. macros
7. files and external variables
8. programming errors

The current values of properties are recorded for each instance of an entity. "Project" properties are those not specifically related to other entities individually or in small combinations—e.g., the overall schedule, and overhead budget and cost entries. The properties of people are generally those that have relevance to a given hypothesis under study—e.g., year of entry into the computing profession, productivity, or current number of simultaneous projects. The most extensive data will be kept on module properties: the source code itself, a short description and interface specification, estimated and current length and complexity, current computed reliability and so forth. For documents, the text and some simple measures such as number of outlined vs. completed sections are maintained. System tests have specifications that go through stages of development; thus their properties are the status of this specification and also the status of the system with respect to passing or failing the test. Macros are centralized source language fragments or forms; depending on the environment, COMPOOL entries or manually inserted COMMON statements serve the same purpose. Macros, files and external variables are usually characterized only by their definition and brief description. Finally, programming errors are "entities" of interest in software research. To go into some detail in this one case, the following properties are typically kept for each error:

1. time found (symptoms appeared)
2. time debugged
3. time fixed
4. classification by type(s)
5. method of discovery (code reading, diagnostic, test, etc.)
6. mental level of error (motor, memory, logic)
7. when made (original code, change, adding instrumentation, etc.)
8. direct costs (man-hours, machine time) of this error.

Relational files are lists that keep track of interrelationships between entities; in effect, they are extensions of the properties of the entities involved. For example, there is a file relating each module to the errors that occurred within it, and another file that keeps track of intermodule references (who-calls-whom).

Event files contain the time and complete details of all transactions that affect the status of the other two files. An event is thus any particular instance of one of the first three major functions—i.e., edit, compile or test. The purpose of the file is mainly historical, to be able to reconstruct a prior state, and also to permit computation of any current state descriptor not originally defined—in other words, to support hindsight research. For practical reasons, very little of this file is kept online.

How Simon operates

Corresponding to each Simon function and each type of transaction used in developing software, a dialog takes place between the user and the system to supply inputs to the data base and to cause requested actions to occur. To illustrate how combinations of data are collected to add to or modify different files during a compilation request, the following sequence of actions might take place:

1. The user is requested to specify a reason for the compilation, e.g., to add instrumentation, or to fix an error. In the case of an error, he may be requested to supply data about the error if it has not been previously recorded, or to cite the error(s) if already entered.
2. A preprocessor scans the source data and adds code for system macros referenced by the module. Other alterations might be made to the code to introduce standard run-time instrumentation. If so directed, the preprocessor also performs analyses of the source code and records the results in the module entity file, e.g., the complexity measure might be calculated, or forbidden forms or references detected.
3. The relational files are updated to show what macros are referenced, and what modules are called by this module.
4. The compiler is called with either a standard set of options or user-specified variations. The object code is directed to the data base. The compiler output is scanned to obtain, for entry into the entity and relational files, the object size, a list of external references set and used, and data about compiler-generated error messages, such as the number and severity of errors or the number of statements flagged.
5. The cost of compilation and current date and time are used to update budget and schedule information in the module entity file.
6. The fact of compilation, and reason therefor, and the time are entered into the online part of the event file; other details (e.g., the compiler-generated listing) are retained in the "paper" part of the event file.

USES OF SIMON

Overview

Research into management techniques is essentially the formulation and validation of hypotheses relating what can be done or seen at one stage of a project to the effects thereof at a later stage. Practical management is largely a matter of applying validated hypotheses, or theorems, to a particular case—although good managers rarely refrain from doing a bit of "research," formal or informal, while they're at it.

In the case of software, we have difficulty formulating hypotheses in the first instance because even the terms are "soft," or ill-defined. For example, it should be possible to relate cost to quality. But we lack understanding of what quality is, and so it is not surprising that we cannot relate cost to quality. Secondly, even widely-held hypotheses are understood only vaguely. For example, everyone seems to agree that modularity is a good thing, and that the lack of it costs money, but we are not agreed on what it is, let alone how to measure it. Many hypotheses, some of them widely publicized, seem to be in this category at present. For example:

1. Structured Programming leads to greater comprehensibility and reliability.
2. Complexity (the inverse of comprehensibility) and the cost of debugging are strongly covariant.
3. The programmer "inherent skill factor," sometimes said to have a 1-to-26 or even higher range, (is/is not) really that important when properly separated from other variables, (for example, learned work habits), and (does/does not) manifest itself in comprehensibility measures which can be taken before reliability can be estimated.
4. Chief-programmer techniques lead to vanishing error rates and lower development costs.

The list is potentially endless. And when it finally comes down to validating any of these hypotheses, even when properly understood, we are up against the same classic problems faced by social scientists: (1) truly controlled experiments of representative scale are impractical, and (2) even just plain observations are lacking or not consistently based.

Simon should overcome this last problem, at least, bringing us closer to the goal of useful measures and theorems in three ways:

1. by hypotheses validation, as mentioned;
2. by consistent data collection, toward the formation of better hypotheses; and
3. to the extent that current hypotheses are correct, by providing a direct visibility aid to the managers and programmers.

Direct use of Simon

As a simple example of how such a system can be a direct aid to the manager, consider what can be learned from the error data described earlier. At any given point, there are a certain number of errors that have been observed, and some smaller number that have been fixed. It is a simple matter to fit a line through the points expressing total number vs. time and another line through the number of corrected errors vs. time and observe whether the lines cross (i.e., all known errors become fixed) before the project deadline. Such a method has been described by Coutinho.[4] Of course, the validity of such a projection will depend on the assumed model of the way in which errors are discovered and removed. The model is an hypothesis that has to be proved, which brings us back to the main purpose of Simon from our standpoint, namely that of serving as a vehicle for measurement definition, hypothesis validation and formation.

SIMON AS A TOOL FOR EXPERIMENT

An experiment

The same data on the number of errors vs. time used in the prior example might be used to evaluate a testing tool or technique by contrasting the plot for modules in which one method of testing was employed with the plot for modules in which it was not used. This is one of the techniques which was used to evaluate FADEBUG-1.[5] All of the quantitative measures of FADEBUG-1 could be derived easily from the Simon data base. In this case, the hypothesis was that a tool such as FADEBUG-1 would reduce the cost of validating software.

Another experiment

We close with a description of an experiment that we plan to conduct with Simon, and that will perhaps summarize its purpose in most concrete terms. The hypothesis to be examined is a refinement of the second one on our list, viz: (a) length of code is roughly proportional to errors of clerical level only; (b) complexity of code (in terms of independent paths, a concept described in Sullivan[6]) is roughly proportional to errors of higher level; and (c) the latter account for so much of the effort expended in debugging that complexity is an approximate indicator of required debugging effort, or when that effort is artificially constrained, unreliability.

1. A software project is selected which will be using the programming environment in which Simon operates.
2. The data collection and analysis requirements for Simon are determined based on the hypotheses. For example, error data must distinguish clerical errors from errors of a higher level, and the time spent debugging an error must be collected from programmers.
3. Any analysis or collection routines not already available under Simon must be written.
4. The system is defined, designed and implemented in the usual way, using Simon as the intermediary between programmers and the computer.
5. The hypothesis is evaluated: complexity does, or does not, correlate well with the higher-level errors and debugging costs (or projected costs to achieve 100 percent reliability) of the several modules. (Of course, there is always the "gray area" of say, 80 percent correlation—which leaves almost 40 percent of the phenomenon to be explained.) Refinements or alternative hypotheses may be developed at this point, fitted to the historical data and further evaluated on other projects.

CONCLUSIONS

Problems of software cost and quality can be solved by providing measures and measure-relating theorems which can be used by managers to plan and control software development. Currently, progress is hampered by the lack of data of the kind and amount needed to validate hypotheses about the definition of such measures, and their utility in controlling cost and quality. A software implementation monitor can aid in the validation of hypotheses and in the development of new insights by making it possible to collect and analyze current data about activities, costs, and product characteristics and their interrelationships.

REFERENCES

1. Hoare, C. A. R., "Computer Programming as an Engineering Discipline," *Electronics and Power*, 9 August 1973, 316-320.
2. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation* 19 (5), May 1973, pp. 48-59.
3. "Design and Construction of an Automated Software Evaluation System," *1973 IEEE Symposium on Computer Software Reliability* April 1973, pp. 28-37.
4. Coutinho, J. De S., "Software Reliability Growth," *1973 IEEE Symposium on Computer Software Reliability*, April 1973, pp. 58-64.
5. Itoh, D., "FADEBUG-I, a New Tool for Program Debugging," *1973 IEEE Symposium on Computer Software Reliability*, April 1973, pp. 38-43.
6. Sullivan, J. E., *Measuring the Complexity of Computer Software*, The MITRE Corporation, Report MTR-2648 Vol. V, 25 June 1973.

Embedded computers—Software cost considerations

by JOHN H. MANLEY

Headquarters Air Force Systems Command
Andrews Air Force Base, D.C.

There is a special subset of the total set of electronic data processors that is growing in importance, especially to the Air Force. This class of computers possesses several unique features that cause software development to be very expensive with relation to the number of lines of code required to make them run. This paper describes some of these unique features and illustrates how they tend to drive software costs upward. Potential solutions to the high cost of software problem in this area are offered which, as the reader will find, are tentative at best since we have only recently begun to investigate in depth the intricacies of what we call "embedded computers."

EMBEDDED COMPUTERS

Embedded computers are information processors which are integral to electromechanical systems such as automated production lines, modern aircraft, ballistic missiles, automated rapid transit systems, naval vessels, spacecraft, and the like. They are considered different than normal computers primarily in the context of how they are built and acquired for a using system. For the purposes of this paper, a computer can be considered to be of the embedded variety when it is:

1. physically incorporated into a larger system whose primary function is not data processing; and
2. integral to such a system from a design, procurement and operations viewpoint.

Furthermore, it is generally not desirable to manage the development and acquisition of an embedded computer and its associated software totally independent of other system parts. Therefore, they are normally acquired as an integrated subsystem or "configuration item" which, in our terminology, represents a well defined part of a larger system.

It is emphasized that this definition of embedded computers prevents them from being classified as "general purpose." The author also feels that they should not be loosely classified as "special purpose" in the generally accepted meaning of this terminology. The fact that many special purpose computer systems exist that are actually unique general purpose machines, such as the STARAN and ILLIAC IV, seems to warrant the use of new terminology for those of the embedded variety. It is the combination of all the descriptive parameters taken together that makes embedded computers unique. Many of the problems with developing the software to make them function are also somewhat different as shall be discussed later.

The argument for partitioning out embedded computers for special consideration may not seem too important to many computer professionals, however it is vitally important that program managers who are responsible for their development and acquisition become more aware of the special problems that embedded computers can create for them, including their impact on the systems that contain them. Special management techniques are necessary for developing embedded computer systems that do not pertain to most general purpose and many special purpose processor systems.

Therefore, much of the following discussion is intended to support the author's firm convictions that:

1. embedded computer systems are sufficiently unique to be placed in a class by themselves; and
2. special management techniques are required to oversee their development, testing, operation and maintenance.

DEVELOPING EMBEDDED COMPUTER SYSTEMS

The systems approach to developing any large scale system can be logically viewed as a consistent method of reducing highly complex problems into sets of relatively simple parts that can be dealt with individually. Subsystems and other "building blocks" or "black boxes" are defined and developed individually. Then, a system architect integrates the parts into a whole. This process permits a single program manager to keep the "big picture" in proper perspective, rather than forcing him to direct his attention to relatively minor aspects of the total system.

When a digital computer subsystem is treated as an individual building block, especially one that is intended to integrate the parts of a large scale system through information transfer, many problems arise in developing its software that cause high costs. The computer system that is embedded into an aircraft in the form of a digital avionics subsystem must receive special attention from the program manager and not be treated simply as another configuration item that can be built in relative isolation from the rest of the system. The importance of this notion can be realized when we consider that an aircraft avionics subsystem includes airborne electronic sub-subsystems such as computers, sensors, display devices, control elements, and so forth; all of which must be integrated by means of electromechanical interface elements to assist the aircrew in managing aircraft resources.

The nature and extent of the problems in creating avionics computer subsystem software are still mostly undefined. For years the use of information generating devices in aircraft subsystems have been largely limited to analog types directed toward semi-autonomous subsystems involving communications, navigation, weapons delivery, electronic countermeasures, stores management, power plant monitoring, and so forth. Very sophisticated analog computers have been developed in the radar navigation and bombardment areas, for example, and routine use of on-board, cathode-ray display devices for monitoring aircraft engine performance have been in existence on transport aircraft for well over twenty years.

Within each of these subsystem areas, however, there are interactions and the consequent need for more efficient information transfer. The logic of using digital devices throughout all subsystems is highly persuasive to satisfy the information transfer and overall integration needs. Therefore, a definite trend has developed toward the replacement of analog with digital equipment. However, the basic point is that this trend has introduced problems that were not foreseen and consequently are still in the process of being defined.

The currently accepted practice used to develop large scale automated systems such as aircraft equipped with digital avionics uses program management techniques. To plan the system development, the overall system is broken down into subsystems, sub-subsystems, and so forth, until a level of detail is arrived at that serves as a baseline specification for design engineers to begin detailed specification of individual parts. At some point in this decomposition process, the program manager's interest is fully satisfied by a detailed description of the inputs and outputs of various components without regard to their internal structure or their detailed interface requirements. The program manager's staff look at a much finer level of detail within their individual functional areas, but still not to the level required by those persons who will actually build the system.

The problem with using this normal systems approach with respect to embedded computer subsystems is that they are treated as a "black box" exactly the same as any other configuration item. In aircraft systems, the tendency is to define the size and weight of the computer hardware so that it will be compatible with the overall aircraft design, weight and balance as early as possible in the planning process. This specification is usually made well before software designers have developed sufficient detail in their area to really know how big a computer is required with respect to core and auxiliary storage. Concurrently, at the macrosubsystem level, tradeoffs are made to reduce the physical size and weight of the computer hardware to its absolute minimum to relieve pressures to develop greater aircraft lift capacities by enlarging the wings or engines. The software problem is greatly increased by these procedures whenever it is later determined that the hardware is inadequate for the software. This forces the use of lower-order basic languages, tricky programming to save space in core, and other devices that all lead to higher software costs.

Therefore, it follows that one of the basic problems involved with embedded computer systems lies in the difficulty of developing them using normal program management procedures. It should also be noted that automatic process control systems and other embedded computer systems, in addition to aircraft digital avionics, are all individually created and require their design specifications to be developed mostly from scratch. At the very least, useful documentation exists for only part of the development problem, and certainly not for the entire task since prior systems are not sufficiently the same to copy, especially in the computer software area. In addition, full complements of automation and computational equipment are generally lacking. It is difficult indeed to find an off-the-shelf computer that can do the job without major modification.

NEED FOR EXTRA CAPACITY

With any large complex automated system such as a spacecraft, early warning and control system, advanced aircraft, and so forth, that contains a controlling embedded computer, it is well understood that regular modifications will be made to add new functions and improve those which were included in the original design. As such modifications are made over time, the software must also be modified, usually requiring additional code. Since the programs in an embedded computer are generally stored in the system itself, the system computer memory can rapidly become saturated. Also, the additional functions often cause a system to become overworked to the point where the cycle time is not fast enough to keep up with the increased workload.

When the memory and speed capabilities are approached through the modification process, it is not normally possible to replace the computer with a more powerful one as can be done relatively easily in a conventional computer center. Since the hardware has been engineered into the overall system, and is therefore relatively fixed, the modifications must be forced to fit through non-standard, "tricky," programming practices which ignore the original strict control standards used during development. This results in generally costly and unreliable software modifications, yet another factor which drives up the life cycle cost of embedded computer systems.

The most logical solution to this problem is to add sufficient extra memory and speed capacity to the original specifications to allow for future modifications. This is an extremely difficult tradeoff for the program manager to make, especially where he is closely scrutinized for his efficiency in reducing costs to an absolute minimum. The recommendation is to educate the program manager and his technical staff in the dangers of cutting the memory and speed specifications too close during initial development. Precisely what is "too close" and how one determines just how much "extra capacity" to allow for is a subject for further research. What is important to point out, is that this is a rather unique problem for embedded computer systems that management must become familiar with to prevent excessive future software costs.

SUBSYSTEM INTEGRATION

Returning now to the avionics area, it was stated previously that many diverse aircraft functions must be coordinated by the embedded computer subsystem. For example, in our military aircraft we must integrate communications, command and control, traffic monitoring, power distribution, flight instruments, flight control systems, detection of an enemy, tracking an enemy, fire control systems, weapons management, and many others. Each of these functions are performed by analog and digital processors that operate on input data supplied by an extremely wide variety of sensors.

During the evolution from stand-alone analog devices to interacting digital systems in our aircraft development, it has been found that many relatively autonomous avionics subsystems are not fully compatible with each other. This has resulted in requirements being placed on the system architect, generally a prime contractor, to force various pieces of the system into being compatible. This practice results in an additional requirement for considerable interface software (and sometimes hardware) development. This adds to the overall complexity of the system software at higher cost, but does not in any way increase overall system capability.

Furthermore, since every newly developed aircraft is substantially different from the last, new mechanical components are acquired that perform substantially the same functions from plane to plane, but are sufficiently different to warrant different software to operate them or to use their developed output data. This is often the case because two processors that develop information from two different segments of the electromagnetic spectrum, for example, such as radar energy and infra-red energy, are usually developed by two different subcontractors. Present procedures do not generally demand investigation to see if these potentially common items could be operated in a redundant mode, with one serving as a back-up system for the other; or whether one could be eliminated by increasing the speed and capacity of the other to the point where it could handle both workloads.

Considerations such as this should be pointed out to the overall system architect by the program manager. We are currently investigating how to prescribe specific systems engineering procedures that will insure that similar items will be thoroughly investigated to see if they can be shared or used to provide redundant back-up capabilities. Such a procedure will ultimately decrease the total cost of software through a reduction in overall system complexity and diversity.

STANDARDIZATION

Since the development of embedded computer systems must be tailored to the function of the overall system they are contained in, standards for system development are taken as those that pertain to the overall system. For example, standards for building railroads would be applicable to an automated rapid transit system, aircraft standards are used for airplanes, and so forth. The difficulty here is that each of these systems use their own methods for developing subsystem software. Thus, even though there are many software development standards available, none are standard for all embedded computer system software. This is not only true across the classes of major systems, but even within classes of systems such as avionics.

Because universal standards do not exist, software developments are often characterized by poor programming practice and inadequate documentation and support software, all of which generally lead to higher than necessary life cycle costs. We are therefore looking carefully at innovations such as structured programming techniques which may be useful to partially standardize coding methodology. We are also in the process of analyzing our configuration control procedures and documentation standards to provide a basis for establishing embedded computer standards by 1975.

The preparation of reprocurement data and technical order handbooks also represents a significant portion of the development and acquisition cost of embedded computer systems. One must consider the usefulness of acquiring reprocurement data in view of its relatively high cost. Alternatives to our militarized versions of "tech orders" are also being examined since it might possibly be more cost effective to use commercial handbooks in some cases. Therefore, the goal of standardization of all data must be carefully evaluated and weighed against what it will cost in the long term.

On the other hand, the standardization of subsystems and components in our automated weapon systems appears to have a large potential for cost savings. Standardization permits reduced numbers of inventory items, such as computer programs, and larger production runs on hardware components. Weapon systems and similar embedded computer systems should take advantage, whenever appropriate, of subsystems on hand or commercially available.

TIME-SHARING

For a fully-integrated system that utilizes an embedded computer for its control, time-sharing becomes an extremely critical problem for software designers. When time-sharing is recognized as a problem in general purpose computer shops, it is frequently because low priority jobs never seem to get into the processor for execution; or users operating remote consoles cannot get timely responses to their inputs. Identical problems in an embedded aircraft computer cannot be tolerated. For example, the central computer must continually poll various sensors to determine whether or not the electrical, hydraulic and other such subsystems are operating correctly. Simultaneously, the computer receives radar and radio sensor signals which must be processed into real time information such as current aircraft position, airspeed, groundspeed, and so forth. The question comes up during software design as to which particular functions are more important than others. Should an impending engine failure take precedence over a radar input indicating the aircraft is flying too close to a mountain? These questions are extremely difficult to answer and generally consume more manhours to study and define than comparable problems in general purpose interactive and time-sharing processors. The net result—increased software costs.

MAINTENANCE AND CONFIGURATION CONTROL

Malfunction analysis in complex embedded computer systems presents additional unique problems which drive up the cost of software maintenance. Whenever mechanical system components exist that are controlled by computer-generated input signals, malfunctions appearing to originate in the mechanical components can also be caused by logic errors in the controlling software. For example, the inability of an automated subway train to stop precisely at the right spot alongside a loading platform might be initially diagnosed as a malfunction of the braking system, sticking relays, or some other mechanical defect. Maintenance technicians would probably take apart and inspect all suspected system components that might be causing the problem as their first order of business. Only after all mechanical possibilities are eliminated would the complex system control software be painstakenly checked for a bug. Since the preliminary detailed mechanical checks that are performed prior to the discovery of a software problem should properly be costed against the software account, the overall software cost for embedded computer systems is uniquely higher insofar as troubleshooting is concerned when compared to other types of computer systems.

In highly complex systems it is virtually impossible to completely test all possible logic paths during software development. Again and again new bugs are discovered whenever the system encounters a set of logical circumstances against which the software must function that had not been anticipated. The occurrence of such bugs in normal management information systems are aggravating since the system might be late producing a particular report, or crash as a worst case. Note carefully that when the software is being used to guide an Air Force bomber close to the ground at the speed of sound, such a bug can be disastrous.

To help reduce the occurrence of bugs in the sensitive control subsystems of major systems, it is necessary to devote considerably more attention to the original planning of the system and, in particular, to the use of a stringent configuration control program. Minor changes to the software during development to make fixes, or to accommodate changes to other system parts, must be carefully analyzed with regard to their impact on other subsystems and the system as a whole.

The real proof of software adequacy and reliability is achieved during the testing phase. Traditionally, software has been tested to determine whether or not it meets the original specifications. Even in operational tests, the major criterion is whether or not it performs to its design. In the case of testing large scale embedded computer systems, it is also necessary to do the following:

1. Perform detailed diagnostic analyses of mechanical components that have failed to determine whether or not software can be a contributing cause.
2. Develop and install automatic test equipment that records the operation of the overall system over time. When any new malfunction occurs, test data will then be available to aid in the diagnosis.
3. If at all feasible, provide one or more systems that are highly instrumented and are capable of duplicating malfunctions that only occur during actual operation and cannot otherwise be observed and analyzed. Such instrumented systems can provide the complex interface data between subsystems that is critical to accurate and timely troubleshooting.

Another major problem with embedded computer software with regard to configuration control is the requirement that it be relatively "tamper-proof." The software that integrates and controls various subsystems in a complex weapon system, for example, must not be made too easily accessible to deter unauthorized personnel from tampering with it. For this reason, the most important programs are generally protected by such methods as storing them in read-only memories. This specialized requirement tends to make software maintenance relatively more costly than in general purpose computer systems since the latter systems' programs can be readily accessed in the computer room. It should also be pointed out here that protection against tampering would probably not be a critical necessity if it were not for the complexity of the integration problem discussed previously.

PROGRAMMER SKILL REQUIREMENTS

Finally, there is one more potential problem area in software development and implementation that may be somewhat unique to embedded computer systems. Although it probably cannot be proved as a fact, it would seem that the basic *modus operandi* of software developers and implementers are a direct result of a specialist's viewpoint and not that of a true systems engineer. Programmers concentrate primarily on getting subroutines to run, making subroutines run together in a program, and getting programs to run precisely in the manner as dictated in the original specifications. The exact methods used to get the programs running to specification (which are frequently more art than science) are centered on the program logic itself and not necessarily on what each fix, patch, or other coding change might do to the overall system.

Although this is speculative reasoning, it might partially explain why many of our key personnel engaged in developing and implementing embedded computer systems keep asking for a "special breed" of programmer, one who is a hybrid between an electrical design engineer and a computer programmer. The experts in this field claim that the normal analyst/programmer does not have the proper outlook on "systems" that he should have. Consequently, the extra awareness must be learned on the job. This ultimately requires additional training expense and, hence, increases the overall cost of special embedded computer system software.

Therefore, we are also beginning to investigate precisely what special skills are required for personnel engaged in developing embedded computer systems. We hope to have some results in this area in the near future, at the very least a determination of whether or not this is a real problem.

SUMMARY

The arguments presented above have attempted to justify the requirement for special management treatment of a class of computers that are "embedded" in large, complex, electromechanical systems. This subset of electronic data processors are more difficult to program, and hence more costly to program, than most other types of computers due to their interactions with other parts of a larger system. More attention to planning their development and stricter configuration control of all modifications and changes are mandatory to prevent adverse side effects in the operation of the overall system due to software bugs. These factors, in addition to the others discussed, cause the overall cost of embedded computer system software to be driven to a level that is proportionately higher than comparable software used in normal data processing applications. For this reason, we are intensively investigating this special subset of computer systems to determine how to improve our acquisition management methods to hold their life cycle costs to a minimum.

An examination of TIC–TAC–TOE like games

by ROBERT C. GAMMILL

The Rand Corporation
Santa Monica, California

INTRODUCTION

There is a class of games which resemble TIC-TAC-TOE. These are games where two players alternately put X's and O's into playing spaces, attempting to get n of their signs in a row. Well-known games of this class are TIC-TAC-TOE (3×3) and QUBIC ($4\times4\times4$). The boards of these games form squares, cubes and hypercubes with n playing positions on a side. We will speak of these games as n^k TIC-TAC-TOE following Citrenbaum.[2]

Many games in this class are trivial, such as TIC-TAC-TOE ($n=3$, $k=2$), while others display considerable character. QUBIC ($n=4$, $k=3$) is an example of one of the latter. No solution is known for the game of QUBIC. It is commercially distributed and a number of computer programs have been written to play it.[1,8] Most of those programs have demonstrated a rather poor understanding of the intricacies of QUBIC.

In the material which follows we will examine a number of results which hold for all n^k TIC-TAC-TOE games. These results demonstrate that QUBIC is the smallest unsolved member of the class. Means will be shown by which a solution attempt is being undertaken. A computer program which plays QUBIC will be described and its use in developing strategies and moving toward a solution will be discussed.

n^k TIC-TAC-TOE GAMES

Some notation will be needed for a concise exposition. Table I gives the needed terms. Some of the terminology (e.g., LOSS) may appear strange unless one accepts our predeliction for defining the board state from X's point of view.

No loss strategy exists

One might think that DRAW STRATEGY should have an alternative meaning when it is a strategy for X, but it turns out that X should not be interested in DRAW for these games. This is because it is known from game theory that, in a finite two-player perfect information game, either the first player has a WIN STRATEGY or both players have a DRAW STRATEGY.[3] In other words, no LOSS STRATEGY exists. This is because if a LOSS STRATEGY existed, X could convert it to his own use and make it a WIN STRATEGY. Thus, to paraphrase the above, between two perfect players who completely understand an n^k TIC-TAC-TOE game, there can be no LOSS. The only possibilities are WIN and DRAW. When a game of this class has been solved, it means that a WIN STRATEGY or DRAW STRATEGY has been produced. Games for which solutions exist are characterized by the type of strategy which has been found. It is well-known that TIC-TAC-TOE is a draw game. In the material which follows we will examine a method developed by Erdös and Selfridge[4] which provides a draw strategy for a large number of n^k TIC-TAC-TOE games. First some examination of the properties of n^k boards is needed.

Board properties

It is important to have a number of equations at hand when examining n^k boards. Equation (1) gives the number of lines in an arbitrary n^k board.

$$L=\frac{(n+2)^k-n^k}{2} \tag{1}$$

Different points in an n^k board will have differing numbers of lines through them. This makes some points more powerful than others. We will need to know the number of lines passing through the most powerful point on the board. When n is odd, the centermost point of the board is the most powerful. Equation (2) gives the number of lines through that center point when n is odd.

$$LCP=\frac{3^k-1}{2} \tag{2}$$

When n is even, there is a collection of most powerful points. The number of lines through these points is given by equation (3).

$$LPP=2^k-1 \tag{3}$$

These equations and many more can be found in Reference 2.

TABLE I

TERM	MEANING
POINT	A playing position where an X or O may be placed.
LINE	A sequence of n points, which when covered by X's will produce win.
X	Pieces played by first player, also his name.
O	Pieces played by second player, also his name.
STATE	A configuration of pieces on the board.
WIN	A state where first player (X) has won.
LOSS	A state where first player has lost (i.e., O wins).
DRAW	A state from which neither WIN nor LOSS can be reached.
STRATEGY	A plan of action for a player.
WIN STRATEGY	A plan (for X) which reaches WIN, no matter what O does.
LOSS STRATEGY	A plan (for O) which reaches LOSS, no matter what X does.
DRAW STRATEGY	A plan (for O) which reaches DRAW, no matter what X does.

Draw strategies

Erdös and Selfridge[4] have devised a strategy by which O may achieve draw under certain conditions. In order to describe the strategy a number of definitions are needed. Table II defines the Erdös-Selfridge value of a line.

TABLE II—Erdös-Selfridge Line Values

Character of line L	V(L)
One or more O's (blocked)	0
empty	1
1 X, no O's	2
2 X's, no O's	4
n X's, no O's	2^n

The value of a board state $V(B)$ is the sum of the values of all lines on the board (4). The value of a move $V(M)$ is the sum of the values of the lines passing through that point, before a piece has been played there (5)

$$V(B) = \sum V(L) \qquad (4)$$

$$V(M) = \sum_{M \in L} V(L) \qquad (5)$$

From the above, the following sequence of equations (6) can be deduced.

$$\begin{aligned} V(B_0) &= \text{number of lines on the board} = \text{initial board value} \\ V(B_1) &= V(B_0) + V(M_1) \\ &= \text{board value after first move (by X)} \\ V(B_2) &= V(B_1) - V(M_2) \\ &= \text{board value after second move (by O)} \qquad (6) \\ V(B_3) &= V(B_2) + V(M_3) \\ &\;\;\vdots \end{aligned}$$

This means that equation (7) holds when j is odd.

$$V(B_{j+2}) = V(B_j) - V(M_{j+1}) + V(M_{j+2}) \qquad (7)$$

If O picks his moves so they always have the maximum possible value (the Erdös-Selfridge strategy) then equations (9) and (10) are true when j is odd. Equation (9) holds because placing an O on a point (M_{j+1}) causes all lines through that point to take value zero. Points contained in those lines may then have reduced value. No point will have increased value. Thus, the value of X's next move $V(M_{j+2})$ cannot be greater than the value of O's move $V(M_{j+1})$.

$$V(M_{j+1}) \geq V(M_{j+2}) \qquad (9)$$

$$V(B_{j+2}) \leq V(B_j) \qquad (10)$$

It is clear that a WIN board state must have a value $V(B)$ which is greater than or equal to 2^n, since at least one line must have n X's in a row. This means that if $V(B_j) < 2^n$ for any odd j, O can prevent X from winning, since all succeeding $V(B_j)$ where j is odd will be no greater.

Finally, if the maximum possible value of $V(B_1)$ is less than 2^n, O can use the Erdös-Selfridge strategy to achieve draw every time the game is played. The maximum value of $V(B_1)$ is the number of lines on the board plus the value of the best possible move. That best move will be the point which has the most lines through it. Thus, using equations (1) through (3) we produce equations (11) and (12), which specify a test for an n^k game being drawn for n odd and n even respectively.

$$\text{(if } n \text{ odd)} \qquad \frac{(n+2)^k - n^k}{2} + \frac{3^k - 1}{2} < 2^n \qquad (11)$$

$$\text{(if } n \text{ even)} \qquad \frac{(n+2)^k - n^k}{2} + (2^k - 1) < 2^n \qquad (12)$$

The smallest interesting game which satisfies these equations is ($n=4$, $k=2$) 4×4 TIC-TAC-TOE. Working out an example for that game is an instructive exercise.

It should be emphasized that this technique not only characterizes complete games, but if at any time the board state value after X's move dips below 2^n, the Erdös-Selfridge criterion declares that draw has been reached.

The Erdös-Selfridge strategy does not apply to TIC-TAC-TOE, i.e., the criterion is not satisfied after the first move. However, it is well-known that TIC-TAC-TOE is a draw game. Games which are known to have draw strategies are summarized in Table III.

Win strategies

Win strategies are known for a number of n^k games. In most cases these games are so trivial that the strategy need not be explained. However, 3^3 TIC-TAC-TOE has a win strategy and a simple explanation may be needed. The initial move should, of course, be in the centermost point, which is a member of 13 lines. Thereafter, it is almost impossible for O

TABLE III—Summary of n^k games

	n, the number of points in each line										
	1	2	3	4	5	6	7	8	9	10	
	1	2	3	4	5	6	7	8	9	10	points
$k=1$	1	1	1	1	1	1	1	1	1	1	lines
	W	ESD	ESD	ESD	ESD	ESD	ESD	ESD	ESD	ESD	strategy
	1	4	9	16	25	36	49	64	81	100	points
$k=2$	4	6	8	10	12	14	16	18	20	22	lines
	W	W	D	ESD	ESD	ESD	ESD	ESD	ESD	ESD	strategy
	1	8	27	64	125	216	343	512	729	1000	points
$k=3$	13	28	49	76	109	148	193	244	301	364	lines
	W	W	W	?	?	?	?	ESD	ESD	ESD	strategy
	1	16	81	256	625	1296	2401	4096	6561	10000	points
$k=4$	40	120	272	520	888	1400	2080	2952	4040	5368	lines
	W	W	CW	?	?	?	?	?	?	?	strategy
E-S number	2	4	8	16	32	64	128	256	512	1024	

W means WIN CW means Citrenbaum WIN
D means DRAW ESD means Erdös-Selfridge DRAW

to achieve draw. Every time X moves, O must block on the opposite side of the cube. If X makes a move that produces two X's in a line in the cube side, he will also probably have produced a line of two X's through the center. The simultaneous production of two lines of two X's results in a win, since O cannot block both.

A theorem by Citrenbaum[2], page 113, states that if an n^k game has a win strategy, then the same strategy will apply for k larger than that. The reasoning is that if a winning strategy can be carried out on a k-dimensional board, it can be carried out in a sub-cube of a $k+1$ dimensional board, etc. The extra dimensions simply allow more space for O to spread his responses over, weakening his defense. Thus, we now know that all $n=3$, $k\geq 3$ games have a win strategy.

Summary of n^k games

In Table III are summarized the results which are known concerning various n^k TIC-TAC-TOE games. From the table it is clear that QUBIC is the smallest game for which no solution exists.

Two interesting conjectures stem from the table.

(a) Conjecture due to Citrenbaum:[2] If $n\leq k$ then the game has a winning strategy. If $n>k$ then it has a draw strategy.

(b) Conjecture due to Gammill: If the number of lines is greater than or equal to the number of points in the board, the game has a win strategy. Otherwise it has a draw strategy.

$$\frac{(n+2)^k-n^k}{2}>n^k \quad \text{implies win}$$

Note, from the table, that this means that Citrenbaum assumes that a draw strategy will be found for QUBIC while the author assumes a win strategy will be found. Both conjectures are supported by all presently known information.

Succeeding sections will describe methods used in the attempt to find a solution (strategy) for the game of QUBIC.

QUBIC

In the preceding sections we have shown that QUBIC is the smallest nontrivial n^k TIC-TAC-TOE game. QUBIC is interesting from a number of points of view. It is a simple enough game to serve as a fruitful test-bed for ideas in strategy analysis. By contrast with chess or checkers, it has such simple rules and structure that the data processing tasks do not overcome the more interesting (and important) analysis tasks. However, despite the simplicity of the game the size of the board state space is sufficient to preclude brute force analysis (64! by unsophisticated methods). Like other sophisticated games (e.g. checkers and chess) it has phases of play which exhibit differing properties and require rather different kinds of analysis. The three phases of QUBIC are represented in the table below.

TABLE OF QUBIC PHASES

PHASE	CHARACTER OF THE PHASE
OPENING	Usually the first 5 to 7 moves. No direct threats occur and no defensive play is necessary. Primary goal is gaining control of board resources (lines and planes).
MIDDLE	Threats and counter threats occur continually. Every move has both an offensive (threat creation) and defensive (threat elimination) component.
END	An unstoppable threat is created by one of the players. The other player may stall, by creating minor threats which must be fended off, but ultimately he is forced into completely defensive play until finally he loses.

In order to examine the analytic tools which are useful in each of the phases, we must examine some specific features of QUBIC.

The QUBIC board

The following facts are known about the board.

ELEMENT	NUMBER
POINTS	64
LINES	76
PLANES	18
RICH POINTS	16
POOR POINTS	48

Rich points are those that are members of seven different lines. Poor points are members of only four lines. Rich points are cube corner and interior points (elements of main diagonals). Rich points capture more lines for a player, so opening play is usually limited to them.

The QUBIC board is large enough so that brute force examination of all possible games is not productive. However, many seemingly different games are actually the same, although rotated, reflected or otherwise transformed. Silver[5] has shown that QUBIC has a group of 192 automorphisms. This includes the 48 standard axis transformations of the cube due to rotation and reflection. Added is a factor of 4 more automorphic images due to three "scrambling" transformations. These transformations do violence to the layout of the cube, but have no effect on the game of QUBIC. Perhaps the best way to describe them is to show how they transform a position. If we describe a position using matrix notation (i, j, k) where the range of indices is 1 to 4, then Figure 1 gives the transformations. For example, using transformation 1, the position $(1, 2, 3)$ becomes $(2, 1, 4)$. Figure

TRANSFORMATIONS

(→ means "changes to")

1	1→2, 2→1, 3→4, 4→3	(inside out exchange)
2	1→4 4→1	(outside exchange)
3	2→3, 3→2	(inside exchange)

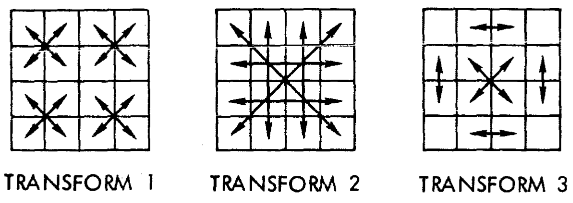


Figure 1—Scrambling transformations applied on a 4×4 square

1 also shows how the transformations look when applied in the 4^2 game. The result of the 192 automorphisms of QUBIC is that on an empty board all the rich points are automorphic images of one another, and likewise for all the poor points. Thus, the first move of the game involves a simple choice, whether to play a rich or poor point.

The opening game

The 192 automorphisms as well as the fact that differing sequences of moves can achieve exactly the same board state cause a dramatic reduction in the number of different board states which are possible in the QUBIC opening. The following table gives the number of distinct states, when play is restricted to the rich points, over the first five moves. The tabulated results were produced by computer enumeration.

TABLE

Comparison of number of states for first 5 moves (on rich points) against number of distinct input sequences.

MOVE	NUMBER OF DISTINCT STATES	NUMBER OF POSSIBLE INPUTS
1	1	16
2	5	240 =15×16
3	20	3360 =14×240
4	103	43680 =13×3360
5	307	524,160=12×43680

It can be seen that the first five moves of QUBIC form a finite automaton of 436 states (when play is confined to rich points). Furthermore, if we are defining a winning strategy for the first player (X), then only the best move from each state where X plays need be defined. This results in only 12 possible states after three moves have been made. Thus, as more becomes known about the middle game, it should be possible to find optimal paths through the opening by a process of enumeration.

The end game

The end game is the best understood aspect of TIC-TAC-TOE games. The end game involves forcing sequences, where one player makes moves which require a specific reply by the opponent to prevent a loss. This sequence ultimately culminates in a win by the forcing player. Figure 2 shows two examples. Example (a) is the simplest form of forcing sequence, requiring only three moves. Example (b) shows a common eleven move forcing sequence. The examples are all planar subparts of the QUBIC board. Forcing situations like that of example (b) are extremely common. Anytime three pieces of one player occupy a plane, without interference from the opponent, it is likely that an eleven move forcing sequence can be found. Despite the importance and frequency of occurrence of this "three-in-a-plane" situation, many QUBIC players (people and computers) do not know about it. A classic example is a game listed in Reference 6 showing a game played between a person and a computer program (written by Daly[1]). In that game the person creates a situation like example (b), the program fails to make a defensive (blocking) move, the person fails to use his forced win, and thereafter the computer forces a win. Regrettably this "double blunder" game has been presented as an example of a skillful computer beating a person at QUBIC. Competitive play, like the ACM Computer Chess Tournaments, appears to be the best way to improve the quality of play.

The force language

Citrenbaum[2] and the writer have independently devised graphical notation systems (languages) for representing and recognizing a forcing situation. Where Citrenbaum has achieved slightly greater generality, creating a notation suitable for representing forcing situations in positional games (of which n^k TIC-TAC-TOE is a subclass), the author has developed a language especially suited to QUBIC. Unlike Citrenbaum's notation, the graphical language to be presented here has been implemented in an efficient program for finding extremely complex forcing sequences. Sequences containing as many as 31 moves have been found. No depth limit is set. The following table presents the elements of the language.

TABLE OF LANGUAGE ELEMENTS

ELEMENT	MEANING
ⓝ	Line containing n X's
□	Unplayed point
→	Membership (point in line)
□→②←□	Line containing 2 X's, showing its two unplayed points

Using these elements we are able to describe the forces which were shown in Figure 2. It is not necessary to show all of the lines in each plane, only those lines directly involved. Figure 3 shows the representation of the forces from Figure 2. The number next to each point indicates the order in which moves occur. Odd numbered moves are X and even moves are O.

From the graphical force descriptions shown in Figure 3 a general characterization of all forces can be produced. To start a force requires at least one line containing two X's. To continue the force requires intersection with a line containing one X, or two intersections with an empty line. Figure 4 provides a graphical description of the configurations which

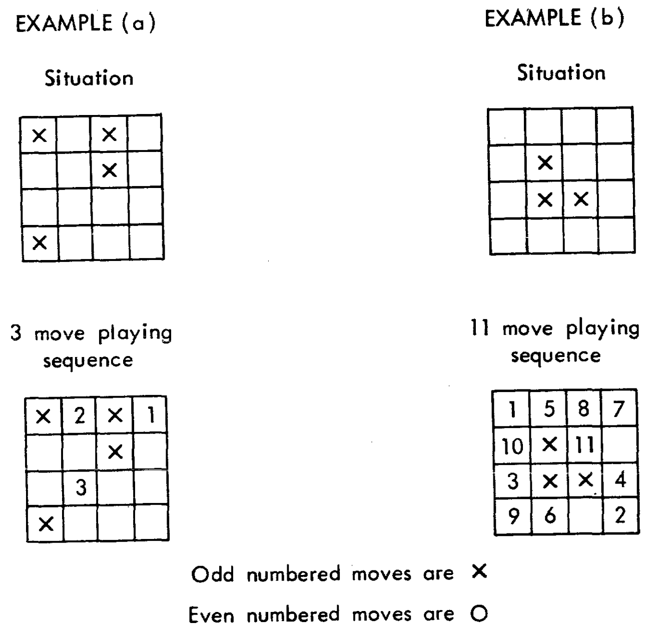


Figure 2—Two examples of forcing situations and the playing sequence leading to WIN

make up forces in part I and examples of some actual forces in part II.

The ability to describe forces in QUBIC provides a powerful new tool. It appears that the language describes all possible forces, but no formal proof has been carried out. An important side effect, not described by the language, is that the force may not work if any of O's replies (even numbered moves) are forced into a line where O has two pieces. This has been termed a counter-force by Gilmartin,[7] because it forces X to defend rather than continue the force. The solution to counter-forces is to avoid them or to order the forcing moves so that the reply which counter-forces is last, allowing the winning move to be made next. Gilmartin has also described an interesting case where forces and counterforces interact in a complex manner, as shown in Figure 5.

The program

A QUBIC playing computer program has been written using the methods described in the preceding sections. The most important features of that program are the following:

(1) A module capable of responding to direct threats, collecting easy wins, and handling other obvious tactical situations.
(2) A module capable of finding and playing out a force, if one exists.
(3) A module capable of evaluating moves in terms of resources (e.g., lines and planes) controlled, and ordering those moves in a (best first) plausibility list. No searching is carried out.

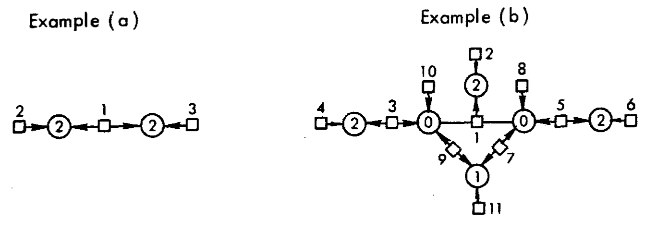


Figure 3—Graphical representation of Figure 2

(4) A module capable of finding forces available for the opponent. This module finds the most plausible move (if one exists) which prevents all forces by the opponent.

This program plays a good game of QUBIC, but of course that statement cannot easily be verified. The surest method of verification is competitive play. The program has beaten many good players, but competition with other good players (computers or humans) is actively being sought.

Perhaps the most interesting facet of the program is that it is much more capable of recognizing board states that contain forcing sequences than is any human being observed

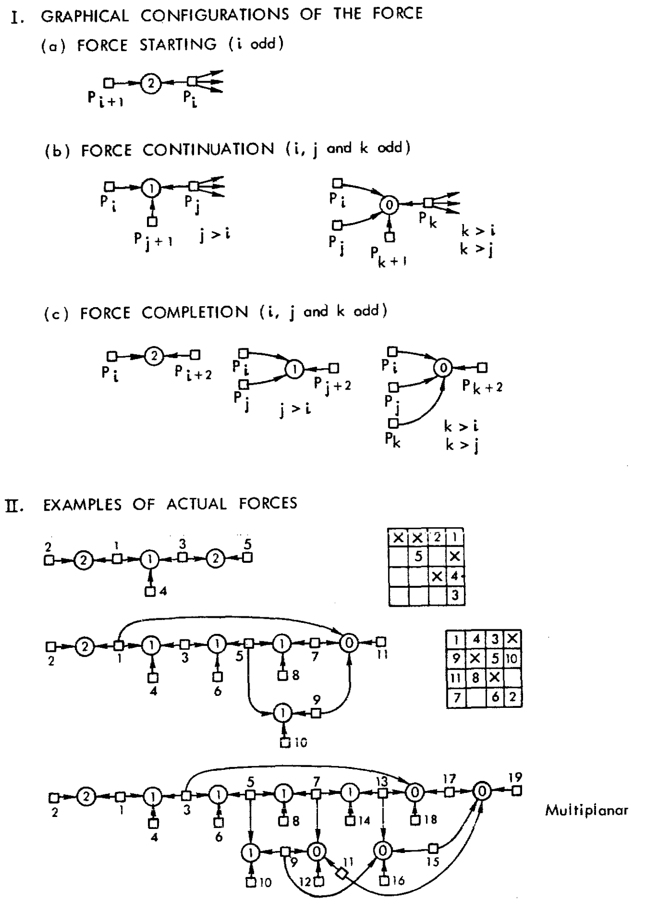

I. GRAPHICAL CONFIGURATIONS OF THE FORCE
(a) FORCE STARTING (i odd)
(b) FORCE CONTINUATION (i, j and k odd)
(c) FORCE COMPLETION (i, j and k odd)
II. EXAMPLES OF ACTUAL FORCES
Multiplanar

Figure 4—Graphical force language and example forces

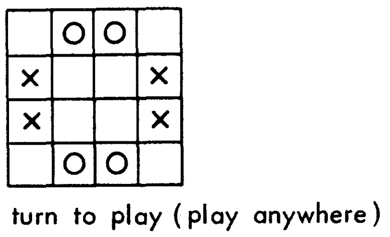


Figure 5—Situation where X wins, but every forcing move produces a counterforce

so far. This has had the interesting result that the program has taught a number of experienced QUBIC players, including the writer, to play a better game. This is accomplished by the module described under (4), which eliminates moves that fail to defend against a force threat. So many of the middle game board states contain forces or force threats, that games between humans invariably contain a number of blunders (normally not as obvious as the game mentioned above from Reference 6). Thus, modules of the program can serve as analysis tools for human players. It is particularly useful, for example, to have the program produce the small set of different moves which will prevent a force by the opponent and for the human to select that move from the set that captures maximum resources or threatens the opponent with a force. Up until recently the program had been organized to act as an independent player (making all its own decisions). As a result of the feeling that in some ways the program has superior capabilities (e.g. in recognizing force states) but in others (e.g. strategic reasoning) the human is better, the program has been reorganized to allow maximum interaction with a human player (i.e., kibitzing).

Where the program formerly would play a game as either X or O, now it may play both sides but accept advice from a human observer. Another mode is for the program to serve as an adviser (to a human player who is playing one or both sides) suggesting sets of moves or warning against certain moves. This can allow the human to act as a QUBIC researcher, and play out a sequence of moves that appears interesting. When the consequence of a particular move sequence has become clear, the game can be backed up to the last point at which X or O may have had a stronger move. A kibitzing or research mode like this might be a valuable addition to computer chess playing programs, as it could allow testing and improvement of the program's capabilities through the posing and solution of sample problems. The use of these research modes of the program have substantially advanced our knowledge of the game of QUBIC. Much work remains to be done, but slowly the game of QUBIC is yielding its secrets. The time when a solution (strategy) can be reported appears to be approaching.

SUMMARY

n^k TIC-TAC-TOE games have been examined and it has been demonstrated that QUBIC (4^3) is the smallest non-trivial member of this class of games. The knowledge and methods used in research toward a solution of the game have been outlined. Finally, a QUBIC playing program which has been modified for use as a research tool has been described.

REFERENCES

1. Daly, William, *Computer Strategies for the Game of Qubic*, Masters Thesis in E.E., MIT, Cambridge, Mass., Jan. 1961.
2. Citrenbaum, Ronald L., *Efficient Representations of Optimal Solutions for a Class of Games*, Systems Research Center Report SRC-69-5, Case Western Reserve University, 1970.
3. Blackwell, D. and M. A. Girshick, *Theory of Games and Statistical Decisions*, Wiley, New York, 1954, p. 21.
4. Verbal Communication with Prof. Andrezej Ehrenfeucht, Dept. of Computer Science, U. of Colorado, concerning unpublished paper by Erdös and Selfridge.
5. Silver, Roland, "The Group of Automorphisms of the Game of 3-Dimensional Ticktacktoe," *American Mathematical Monthly*, Vol. 74, 1967, pp. 247-254.
6. Slagle, James R., *Artificial Intelligence: The Heuristic Programming Approach*, McGraw-Hill, New York, 1971, pp. 38-39.
7. Gilmartin, Paul, *An Efficient Algorithm for Detecting Forces in QUBIC*, Department of Computer Science, University of Colorado, January 1973 (unpublished paper).
8. King, Paul F., *A Computer Program for Positional Games*, Report 1107, Jennings Computing Center, Case Western Reserve University.

Provable programs and processors*

by DONALD I. GOOD**

USC/Information Sciences Institute
Marina del Rey, California

and

The University of Texas at Austin
Austin, Texas

INTRODUCTION

"A proof of correctness guarantees that a program will run correctly every time it is executed." That statement is not necessarily true. Suppose, for sake of concreteness, that a valid proof of a Fortran program has been constructed. When this program was proved, it most likely was proved in isolation from the other software components which ultimately will be involved in actually making the program run. So, even though we have proved the Fortran program, one of these other components, or the system hardware, may malfunction causing the actual machine language program that is executed to produce an error. These comments are not an argument against proving programs at the Fortran level, but rather an indication of the eventual need for a completely proved computing system.

Although, certainly, the development of even a small scale system that is completely proved must be viewed as a long term goal, the Nucleus project described here is intended to be a small, initial step in that direction. This paper summarizes the project and evaluates the progress made so far.

THE NUCLEUS LANGUAGE

The Nucleus project revolves around the Nucleus programming language as described in Reference 1. Nucleus was designed with three major considerations in mind, program provability, processor provability, and expressability of non-trivial programs. These considerations were manifest in seven specific design goals.

1. Program Provability. Every program in the language must be provable by the inductive assertion method.
2. Structured Programs. The language should support the ideas of structured programming.
3. Provability of a Verification Condition Generator. It must be possible to prove the correctness of a verification condition generator for Nucleus.
4. Provability of a Compiler. It must be possible to prove the correctness of a Nucleus compiler.
5. Ease of Compilation. Nucleus programs must be easily compilable into almost any machine language.
6. Rigorous Language Definition. All aspects of Nucleus, both syntactic and semantic, must be defined rigorously.
7. Express Non-trivial Programs. Nucleus must be capable of expressing non-trivial programs such as its own verification condition generator and compiler.

The central problem in designing Nucleus was in balancing the first six design goals against the seventh, expressibility of non-trivial programs. Ultimately, the principle was followed of making the language just powerful enough so that the Nucleus verification condition generator and compiler could be written in Nucleus without a great deal of difficulty. The following is a very brief summary of the language.

Essentially, the structure of a Nucleus program is the declaration of global variables followed by the declaration of procedures. Procedures are recursive, but have neither parameters nor local variables, and there is no block concept. Every procedure has unrestricted access to every global variable, and those are the only variables to which the procedures have access. This simple, but very primitive method for accessing program variables was motivated by several considerations. First, the construction of verification conditions is somewhat simplified if the variable X always refers to the same data object whenever it appears in an inductive assertion, and simpler construction leads to a simpler proof of the verification condition generator. Second, certain problems in proving procedures with parameters had been pointed out by Hoare.[2] Ultimately, the choice was made to avoid these problems rather than to solve them. Also, since there are no parameters and no local variables, the compiler also would be considerably simpler, and hence, easier to prove.

* This research was supported in part by NSF grant GJ-36424 and by the Advanced Research Projects Agency under Contract No. DAHC 15-72-C-0308, ARPA Order No. 2223/1, Program Code No. 3D30 and 3P10.

** Views and conclusions contained in this study are the author's and should not be interpreted as representing the official opinion or policy of the University of Southern California or any other person or agency connected with it.

This rather drastic decision about parameters and local variables now certainly seems to be overly restrictive. In particular, Hoare[2] and Hoare and Wirth[3] describe ways that can be used to construct verification conditions for the types of parameter passage found in Pascal, and also indicate ways of treating local variables. It is not yet clear, however, how much these features would complicate the proofs of the verification condition generator and compiler.

The types of data objects that can be declared in Nucleus are quite limited. There are three primitive data types INTEGER, BOOLEAN, and CHARACTER. A value of type CHARACTER is just a single character in the basic character set of the language, these values being used primarily in input/output operations. In addition to simple variables, there is only one other kind of data structure, singly subscripted arrays. An array is declared with a constant upper bound and has an assumed lower bound of zero. Nucleus was limited to these simple data objects because, essentially, these were the only objects for which program verification methods were known. Since that time some additional progress on more sophisticated data objects has been made by Burstall[4] and Hoare.[5] The restriction to only singly subscripted arrays was not dictated by verification methods, but was chosen purely for the sake of simplicity. This eventually proved to be a bad decision because this makes it quite awkward to use one of the favorite tools of systems programmers, the table. This problem became readily apparent when faced with writing the verification condition generator and compiler in Nucleus. Allowing at least doubly subscripted arrays would help greatly in writing useful Nucleus programs and would cost very little in terms of additional verification technique.

The statements available in Nucleus are much more sophisticated than the variable accessing method and data objects. The fundamental statement of the language is the assignment statement which is unique only with respect to implicit fault conditions in expression evaluation, divide and modulo by zero, integer overflow, and array subscript violation. If any of these conditions occur, the program halts.

The control statements in the language were influenced strongly by the ideas of structured programming. In addition to a recursive procedure call, statements of Nucleus include

```
IF ⟨exp⟩ THEN ⟨stmtlist⟩ FI
IF ⟨exp⟩ THEN ⟨stmtlist⟩ ELSE ⟨stmtlist⟩ FI
WHILE ⟨exp⟩ DO ⟨stmtlist⟩ ELIHW
CASE ⟨exp⟩ OF ⟨alternativelist⟩ ESAC
CASE ⟨exp⟩ OF ⟨alternativelist⟩ ELSE ⟨stmtlist⟩ ESAC
```

where an ⟨alternativelist⟩ is a ⟨stmtlist⟩ preceded by a list of integer labels. In addition to the preceding statements, Nucleus also has a restricted GOTO. The GOTO can jump to any point within a procedure, but it cannot cross a procedure boundary. Under this restriction, the GOTO does not cause any technical problems in constructing verification conditions.

The statements that were absolutely essential to have in Nucleus in order to write a realistic verification condition generator and compiler, but for which verification techniques were not known, were simple READ and WRITE statements. The form of the read statement is READ arrayname, and its function is to transfer the next input record into the array. Element zero of the array is set to indicate whether the record read was, or was not, an eof (end-of-file) record. If the record is not eof, the rest of the array, beginning at element one, is filled with the characters of the record. If the record is eof, the rest of the array is not changed. The WRITE statement behaves in a similar way.

The one statement that was put into Nucleus strictly for purposes of provability was the ASSERT, which is the means by which inductive assertions are embedded in the program. ASSERT statements have no effect on the operation of the program and are used only in proving its correctness. In appearance, ASSERT is like the ordinary Algol comment, except that the word ASSERT is used instead of COMMENT. This gives the ASSERT statement a very flexible range of expressive power, which allows one to state conveniently the complex properties of the verification condition generator and compiler that eventually have to be proved. This type of extremely loosely formed assertion is adequate for the construction of verification conditions because the construction process requires only a way of detecting program variables in assertions, and this can be done with the help of the variable declarations at the beginning of the program. However, if the verification conditions are to be proved automatically, a more precise assertion language must be used.

PROVABILITY OF NUCLEUS PROGRAMS

For the most part, features were included in Nucleus only if it was well-known how to construct verification conditions for them; the notable exceptions begin the input/output statements, procedures with side effects, and the fault conditions in expressions. This section gives examples of how these features can be handled. More detailed descriptions can be found in References 6 and 7.

First, consider the read statement in the READINTEGER procedure of the following program. The statement

```
CHARACTER ARRAY INLINE[80];
INTEGER X,Y,Z,INTVAL,I;
PROCEDURE ADDXANDY;

ASSERT IF P IN [1,2] THEN K(P) IN [1,10];
ASSERT IF P IN [1,2] THEN NOT :REOF(P);
ASSERT IF P IN [1,2] AND Q IN [1,K(P)]
        THEN :RDFL(P,Q) IS A DIGIT CHARACTER;
```

```
ASSERT IF P IN [1,2] THEN :RDFL(P,K(P)+1) IS A BLANK;
ENTER READINTEGER;
X := INTVAL;
ENTER READINTEGER;
Y := INTVAL;
Z := X + Y;
ASSERT Z = DECINTVAL(:RDFL(1,[1,K(1)])) +
           DECINTVAL(:RDFL(2,[1,K(2)]));
EXIT;
PROCEDURE READINTEGER;
ASSERT NOT :REOF(:RDHD+1);
ASSERT IF Q IN [1,K(:RDHD+1)]
         THEN :RDFL(:RDHD+1,Q) IS A DIGIT CHARACTER;
ASSERT :RDFL(:RDHD+1,K(:RDHD+1)+1) IS A BLANK;
READ INLINE;
I := 1;
INTVAL := 0;
ASSERT IF J IN [1,80] THEN INLINE[J] = :RDFL(:RDHD,J);
ASSERT IF Q IN [1,K(:RDHD)]
         THEN INLINE[Q] IS A DIGIT CHARACTER;
ASSERT INLINE[K(:RDHD)+1] IS A BLANK;
ASSERT :RDHD = :RDHD.0 + 1;
ASSERT I IN [1,K(:RDHD)+1];
ASSERT INTVAL = DECINTVAL(:RDFL(:RDHD,[1,I-1]));
WHILE INLINE [1] NE " DO
  INTVAL := INTVAL * 10 + (INTEGER(INLINE[I]) - 27);
  I := I + 1;
ELIHW;
ASSERT INTVAL = DECINTVAL(:RDFL(:RDHD,[1,K(:RDHD)]));
ASSERT :RDHD = :RDHD.0 + 1;
EXIT;
```

READ INLINE of procedure READINTEGER reads the characters of the next record of the standard input file into the array elements INLINE[1], . . . , INLINE[80]. It is assumed that this next record is not an eof record, that there is a continuous sequence of digit characters beginning in column one of the record, and that this sequence is followed by a blank. These assumptions are stated explicitly in terms of the "system" variables, :RDHD, :RDFL, and :REOF, in the three assert statements at the beginning of the procedure. :RDHD is the "read head" variable which always equals the number of the last input record read and is advanced automatically by the read statement. :RDFL(r,c) and :REOF(r) are two functions that define the standard input file. :REOF(r) is a boolean-valued function that is TRUE if record r is an eof record and FALSE if not. For records that are not eof, :RDFL(r,c) is the single character in column c of record r. In the assertions, :RDFL(r,[a,b]) denotes the sequence of characters in columns a through b, and K(r) is an auxiliary function that is assumed, and used, strictly for stating the properties to be proved. The effect of the READINTEGER procedure is described by the two assertions at its exit. Upon exit, INTVAL is equal to the decimal integer value of the digit character string beginning in column one of the last record read. Also, the value of :RDHD, upon exit, is equal to its value upon entry, :RDHD.0, plus one.

The verification condition for the path involving the read statement is

```
0.1  NOT :REOF(:RDHD+1)
0.2  IF Q IN [1,K(:RDHD+1)] THEN :RDFL(:RDHD+
       1,Q) IS A DIGIT CHARACTER
0.3  :RDFL(:RDHD+1,K(:RDHD+1)) IS A BLANK
0a   :REOF(:RDHD+1) IMPLIES INLINE.1[0] = "T
       AND [1 LE $ LE 80 IMPLIES
            INLINE.1[$] = INLINE[$]]
0b   NOT :REOF(:RDHD+1) IMPLIES INLINE.1[0]=
       "F AND [1 LE $ LE 80 IMPLIES
               INLINE.1[$] = :RDFL(:RDHD+1,$)]
       AND [81 LE $ LE 80 IMPLIES
            INLINE.1[$] = INLINE[$]]
0c   :RDHD.1=(:RDHD)+1
1    I.1=1
2    INTVAL.1=0
- - - - - - - - - - - - - - - - - - - - - - -
3.1  IF J IN [1,80]
       THEN INLINE.1[J] = :RDFL(:RDHD.1,J)
3.2  IF Q IN [1,K(:RDHD.1)] THEN INLINE.1[Q] IS A
       DIGIT CHARACTER
3.3  INLINE.1[K(:RDHD.1)+1] IS A BLANK
3.4  :RDHD.1 = :RDHD + 1
3.5  I.1 IN [1,K(:RDHD.1)+1]
3.6  INTVAL.1 = DECINTVAL(:RDFL(:RDHD.1,
       [1,I.1-1]))
```

This verification condition is read as an implication with the dashed line (— — —) corresponding to the implies operator. The antecedent is the conjunction of the lines above the operator, and the consequent the conjunction of the lines below it. The ".1" that is appended to several of the variables is an "alteration counter." Each time a variable is altered, its alteration counter is increased by one in order to distinguish the different values it may attain as the path covered by the verification condition is traversed.

The verification condition term for the read statement is the conjunction of the lines labelled 0a, 0b, and 0c. Line 0a describes the effect of the read if the record read was an eof, and line 0b describes what happens if the record read is not an eof. Line 0c describes the automatic incrementing of :RDHD. In this particular verification condition, line 0.1 affirms the antecedent of line 0b, thus giving the part of the consequent of line 0b that is needed to prove the consequent lines of the entire verification condition.

The treatment of Nucleus procedures is illustrated by the single verification condition of procedure ADDXANDY which consists of two calls of procedure READINTEGER.

```
0.1       IF P IN [1,2] THEN K(P) IN [1,10]
0.2       IF P IN [1,2] THEN NOT :REOF(P)
0.3       IF P IN [1,2] AND Q IN [1,K(P)] THEN
              :RDFL(P,Q) IS A DIGIT CHARACTER
0.4       IF P IN [1,2] THEN :RDFL(P,K(P)+1) IS A
              BLANK
0(PRV)    NOT :REOF(:RDHD+1)
0(PRV)    IF Q IN [1,K(:RDHD+1)] THEN :RDFL
              (:RDHD+1,Q) IS A DIGIT CHARACTER
0(PRV)    :RDFL(:RDHD+1,K(:RDHD+1)) IS A
              BLANK
0         INTVAL.1 = DECINTVAL(:RDFL(:RDHD.1,
              [1,K(:RDHD,1)]))
0         :RDHD.1 = :RDHD + 1
1         X.1=INTVAL.1
2(PRV)    NOT :REOF(:RDHD.1+1)
2(PRV)    IF Q IN [1,K(:RDHD.1+1)] THEN :RDFL
              (:RDHD.1+1,Q) IS A DIGIT CHARACTER
2(PRV)    :RDFL(:RDHD.1+1,K(:RDHD.1+1)) IS A
              BLANK
2         INTVAL.2 = DECINTVAL(:RDFL(:RDHD.2,
              [1,K(:RDHD,2)]))
2         :RDHD.2 = :RDHD.1 + 1
3         Y.1=INTVAL.2
4         Z.1=X.1+Y.1
— — — — — — — — — — — — — — — — — — — — — —
5.1       Z.1 = DECINTVAL(:RDFL(1,[1,K(1)])) +
              DECINTVAL(:RDFL(2,[1,K(2)]))
```

The parts of this verification condition due to the first procedure call are the three lines labeled 0(PRV) and the two following lines labelled just 0. The three PRV lines are derived from the initial assertions of the called procedure, and the PRV indicates that they are to be proved (from the lines preceding them) rather than assumed. These lines are formed by inserting the current values (with respect to the traversal of the path) of the program variables into the initial assertions of the called procedure. Thus, in proving these lines, we are proving that the initial assumptions of the procedure are satisfied at this particular call. The lines labelled just 0 are formed from the exit assertions of the called procedure. We are entitled to assume these by virtue of the fact that we have proved the initial assumptions of the procedure and the assumption that, eventually, the called procedure will be proved with respect to its initial and final assertions. The final assertions of the called procedure, however, must be stated with respect to the values of the program variables upon exit from the called procedure. This can be done simply by the use of alteration counters. The called procedure is examined to determine the set of variables that potentially could be altered during the course of its execution. For READINTEGER, these variables are INLINE, :RDHD, I, and INTVAL. The alteration counter of each of these potentially alterable variables is increased by one across each call of the procedure. The counters of variables that cannot be altered by the procedure, such as X, Y, and Z, are left unchanged. Then in deriving the lines from the exit assertions of the procedure, the updated alteration counters are used for exit values of the program variables. Notice, for example, the different treatment of INTVAL and X across the two procedure calls in the example.

Fault conditions in evaluating expressions are handled by the same technique used to prove that the initial assumptions of called procedures are satisfied. For example, if an expression has a division, a PRV line will be constructed that requires that the denominator not equal zero.

METHOD OF DEFINITION

Probably the most unique feature of Nucleus is the method used to define it. The syntax of Nucleus is defined by two separate transition networks, one defining the parser, and the other the scanner. Both networks use the extensions described by Woods[8] which allow the recognition of context-sensitive languages. In addition to defining the syntax, the parsing network also defines a translation from Nucleus programs into code for an abstract machine. The operation of the abstract machine, then, is defined by axioms as suggested by Burstall.[9] This method was a major contributing factor to the provability of the verification condition generator. A more complete description of the method than the one given below can be found in Reference 1.

The networks used to define the Nucleus parser and scanner follow closely the ideas of the "augmented transition network grammars" described by Woods[8] for dealing with natural languages. The basic idea is that a set of programmable registers is associated with the network, and each transition arc in the network is allowed to have a test and a sequence of actions defined on the registers. The test, is considered to be part of the transition condition from one state to the next, and when a transition is made, any actions that are associated with the transition arc are performed. This allows the network to specify the context-sensitive features that exist in most actual programming languages. For example, frequently it is required that an identifier cannot be referenced before it is declared and this type of constraint usually is handled in some ad hoc way. This constraint can be specified quite easily using the tests and actions of the augmented networks. For example, in recognizing the declarations of a Nucleus program, actions may set a register, say IDLIST, to be the list of all identifiers that are declared. Then, in the part of the network that recognizes Nucleus statements, each time an identifier is found, a test is made to see if it is in IDLIST. Thus, the augmented networks provide a uniform mechanism for defining completely the context-sensitive syntax of Nucleus, and ad hoc rules are not needed. This is extremely helpful in proving the correctness of the language recognition parts of the Nucleus generator and compiler.

In part, the semantics of Nucleus are defined by the parsing network which also specifies a translation from Nucleus programs into a set of well-formed sentences in the predicate calculus. For example, the sentences corresponding to the declarations and the READINTEGER procedure of the program given above are

```
ARRAY(INLINE,80)
SIMPLE(X)  SIMPLE(Y)  SIMPLE(Z)
  SIMPLE(INTVAL) SIMPLE(I)
READ(0,INLINE)
ASSIGN(1,I,1)
ASSIGN(2,INTVAL,0)
IF(3,INLINE[I] NE "     ,4,7)
ASSIGN(4,INTVAL,(INTVAL*10)+
  (INTEGER(INLINE[I]—27)))
ASSIGN(5,I,I+1)
JUMPTO(6,3)
EXIT(7)
```

(Some of the details of these sentences have been omitted to make them more illuminating than intricate.) These sentences are what Burstall calls the "structural description" of the program. The semantic axioms, then, are defined in such a way that the execution of the program can be deduced, by the ordinary rules of logic, from the structural description. In the Nucleus definition, the structural description may be viewed as a program for an abstract machine, and the axioms as defining the interpreter for that machine.

The execution of a Nucleus program is defined as a function E from program steps (non-negative integers) into state vectors—that is, the execution is a sequence of state vectors E[0], E[1], ... Each state vector in the sequence is itself viewed as a function from names into values. Each state vector has in its domain the names of the declared variables as well as certain "system" variables such as :RDHD and a location counter called :LOC. From a function such as SIMPLE(X) in the program above, it can be inferred, from the axioms, that X is a name in the domain of every state vector in the execution. Since each state vector is a function, the value of X in, say, state vector four is E[4](X).

In addition to the "declarative" axioms that define the domain of the state vectors, there are also "evaluative" axioms that describe precisely how expressions are evaluated, including the fault conditions. A third set of "imperative" axioms define the transition from one state vector to the next. These imperative axioms define the effect in the abstract program of the functions such as READ, ASSIGN, IF, JUMPTO, and EXIT. These functions are interpreted as the abstract instructions of the program with the first argument of each function being regarded as the abstract address that contains the instruction. The imperative axioms also define precisely the conditions of program termination.

IMPLEMENTATION

It is clear that it is fairly straightforward to implement Nucleus from this type of language definition. Such an implementation in Pascal for the CDC 6600 is described in Josue.[10] This implementation is a two pass compiler that corresponds very closely to the Nucleus definition. Pass I simulates the transition networks and translates the Nucleus program into its corresponding abstract program. This part of the compiler actually was used to help debug the translation defined by the networks. Pass II then compiles machine language from the abstract program. Although not done, it would also be a simple matter to write an interpreter for the abstract program which would correspond fairly directly to the semantic axioms.

This method of definition also leads to an interesting technique for debugging the semantics of the language. In principle, the entire execution of the program should be deducible from the abstract program and the axioms. Therefore, one way of debugging the axioms and the translation into the abstract program is to do the translation and then begin making deductions from the program based on the axioms. (Although it has not been done in any of the work on Nucleus so far, this could be done by an automatic theorem prover.) In using this deduction technique on Nucleus, semantic bugs often were detected in one of two ways. The most common bug was one of inconsistency in which an abstract program and the axioms led to a contradiction. This usually was caused by a multiple definition of some component of a state vector. The other common error was one of incompleteness. In this case some point in the execution would be reached from which it was not possible to deduce the next state vector.

The abstract program also can be used as the basis for driving a verification condition generator. Such a generator was implemented by Wang[7] in Snobol 4 on the CDC 6600. This generator simply follows control paths through the abstract program, and constructs verification conditions using alteration counters much like the ones shown in the examples above.

An interesting extension was made to the Wang generator to permit simultaneous top down design and proof of programs. The generator was extended to allow a procedure to have entry and exit assertions, but to have its body specified as PENDING. Verification conditions are not generated for pending procedures, but they can be called. Since the entry and exit assertions of the pending procedure are present, the proper verification conditions can be constructed for their calls if it were known what program variables the pending procedure might alter. The generator takes a totally pessimistic view and assumes that the pending procedure potentially changes the value of every program variable. This means that the alteration counter of every program variable is increased across a call of a pending procedure. The effects of this quite drastic assumption can be nullified, however, by explicitly stating in the exit assertion of the pending procedure that whatever variables are of interest are not changed. With the pending feature, it is possible to do, in parallel, a top down design and proof of a program. The top level procedure can be written, and the second level ones specified as pending. At this point the top level procedure can be proved, and then we can proceed to expand and prove the second level procedures.

PROVABILITY OF THE NUCLEUS PROCESSORS

A proof of a Nucleus verification condition generator, written in Nucleus, is described by Ragland.[6] This generator also follows the Nucleus definition very closely and consists of two parts. The first is a simulation of the transition network which does the syntactic analysis and translates Nucleus programs into abstract programs, and the second part constructs verification conditions from the abstract program. A proof of a Nucleus compiler has not yet been undertaken, but the first part of the Ragland generator could be used, in tact, as the first pass of a two pass compiler so that the only part that would remain to be proved would be the code generation phase.

The two parts of the generator are reflected in its proof. The transition network simulation part is proved using an equivalence proof, and the part that actually constructs verification conditions is proved using inductive assertions. The contribution of the transition network definition of Nucleus to the practical provability of the generator can not be overestimated. Since the networks give a procedural definition of the syntax of the language, and also of the translation into abstract programs, it was a relatively straightforward matter to show, on a segment by segment basis, that the simulation in the first part of the generator is equivalent to the defining network.

The proof of the second part of the generator is done by the conventional inductive assertion method. As a prelude to this proof, verification conditions were defined for all the possible semantic constructs in Nucleus, and it was proved, by hand, from the semantic axioms, that these verification conditions are valid. The inductive assertion proof of part two of the generator then shows that these valid verification conditions are written onto the standard output file.

This proof also is worthy of note because of the size of the generator. The entire generator consists of 203 Nucleus procedures, most of these occupying less than one page including assertions. Of these 203 procedures, 103 were proved by equivalence proofs, and 100 were proved using inductive assertions. One of the most encouraging aspects of this large proof was that, on the basis of subjective evaluation, 70 percent of the things that were proved were quite simple, and it appears that almost all of these could have been proved mechanically.

BOOTSTRAPPING CORRECTNESS

The ultimate purpose of Nucleus is to serve as the bottom level of a bootstrapping process for developing correct processors for other more sophisticated languages. The central idea is to develop a correct verification condition generator and compiler for Nucleus and use these to build processors for other languages. The key question, however, is how to construct the first correct Nucleus generator and compiler that are to serve as the basis for further bootstrapping. The claim is that an unproved generator, such as the one written by Wang, can be used properly to help build a proved generator, such as the one written by Ragland.

The Wang generator, extended with the PENDING feature, was used heavily in proving the Ragland generator. The key point is that it is not important to the Ragland proof that the Wang generator work correctly on all Nucleus programs. What is important, is that it work correctly on one particular program, the Ragland generator. If the unproved generator does work correctly on that one program, then the proper verification conditions are constructed for the Ragland generator, and thus, its correctness depends strictly, and properly, on the validity of the proofs of the verification conditions. This clearly, then, raises the question of how one verifies that the unproved generator worked correctly on that one particular program. This was done, very carefully, by the standard debugging technique of comparing the program input and output.

This development of a proved generator from an unproved one leads to the observation of an interesting relation between conventional debugging by test cases and proofs of correctness with respect to developing program verifiers. First, note that the same technique above would apply if the Ragland generator had been used to assist in its own proof rather than the Wang generator, and also that the technique still applies if we use complete program verifiers rather than just verification condition generators. We now observe the following property: if a program verifier operates correctly just one time, using itself as input, and obtains a proof, then that verifier operates correctly for all input programs. Said more simply, if it works on one input, it works for all inputs. This relationship is discussed somewhat more fully in Good[11] with respect to both verifiers and compilers.

CONCLUSION

With the exception of a proof of a Nucleus compiler, all of the design goals of Nucleus have been attained. Although this proof has not yet been attempted, the proof of the Ragland verification condition generator strongly indicates the feasability of the compiler proof. In meeting the design goals of Nucleus new techniques have been developed for proving programs with input/output statements, procedures with side effects, and fault conditions in expression evaluation. The main contributor to the attainment of the design goals was the language definition method consisting of transition networks and axioms. This method particularly contributed to the practical provability and easy implementation of the Nucleus processors.

The main thrust of the Nucleus project has revolved around the design of the Nucleus language so that the language could express non-trivial programs, we could prove those programs, and also process those programs with correct processors. Ultimately, the important programs to be written, and proved, in Nucleus are processors, such as verifiers and compliers, for other more sophisticated languages. When this goal is attained Nucleus will have served its purpose.

REFERENCES

1. Good, D. I. and L. C. Ragland, *"Nucleus—A Language of Provable Programs,"* In *Program Test Methods*, Hetzel, W. C. (Ed.), Prentice Hall, 1973.
2. Hoare, C. A. R., "Procedures and Parameters: An Axiomatic Approach," In *Symposium on Semantics of Algorithmic Languages*, Engeler, E. (Ed.), Springer-Verlag, 1971.
3. Hoare, C. A. R., and N. Wirth, "An Axiomatic Definition of the Programming Language Pascal," *Berichte der Fachgruppe Computer-Wissenschaften*, 6, E. T. H., Zurich, November 1972.
4. Burstall, R. M., "Some Techniques for Proving Correctness of Programs which Alter Data Structures," In *Machine Intelligence 7*, Meltzer, B. and Michie, D. (Eds.), John Wiley and Sons, 1972.
5. Hoare, C. A. R., "Proof of Correctness of Data Representations," *Acta Informatica* 1, Springer-Verlag 1972.
6. Ragland, L. C., *A Verified Program Verifier*, Ph.D. Thesis, The University of Texas at Austin, June 1973.
7. Wang, Y. L., *A Nucleus Verification Condition Compiler*, Masters Thesis, The University of Texas at Austin, June 1973.
8. Woods, W. A., "Transition Network Grammars for Natural Language Analysis", *Communications of the ACM*, 13, 10, October 1970.
9. Burstall, R. M., "Formal Description of Program Structure in First Order Logic," In *Machine Intelligence 5*, Meltzer, B. and Michie, D. (Eds.), American Elsevier, 1970.
10. Josue, E. V., *The Nucleus Compiler*, Masters Thesis, The University of Texas at Austin, June 1973.
11. Good, D. I., "Developing Correct Software," In *Proc. of the First Texas Conference on Computer Systems*, June 1972.

A language-independent programmer's interface

by ROBERT M. BALZER

USC/Information Sciences Institute
Marina del Rey, California

INTRODUCTION

This paper addresses the general problem of creating a suitable on-line environment for programming. The amount of software, and the effort required to produce it, to support such an on-line environment is very large relative to that needed to produce a programming language, and is largely responsible for the scarcity of such programming environments. The size of this effort was largely responsible for the scrapping of a major language (QA4[1]) as a separate entity and its inclusion instead as a set of extensions in a LISP[2] environment. The few systems which do exist (e.g., LISP, APL,[3] BASIC,[4] and PL/I[5]) have greatly benefited their users and have strongly contributed to the widespread acceptance of the associated language.

At a bare minimum, a suitable programming environment consists of an on-line interpreter (or incremental compiler), an integrated interactive source-level debugging and editing system, and a supporting file structure. More extensive environments would include such facilities as automatic spelling correction, structural editors, tracing packages, test case generators, documentation facilities, etc.

Looking at several programming environment systems, one recognizes much uniformity. Most of the software supporting these systems is similar in both its organizational structure and functions. The systems differ in detail more from style differences between the system designers than from differences required by the programming languages.

The Programmer's Interface (PI) concept attempts to exploit this uniformity by creating a single programming environment capable of easily interfacing users with a wide variety of on-line programming languages. Users would then have the full facilities of this environment at their disposal. The PI is thus responsible for transforming these programming LANGUAGES into SYSTEMS. The cost of providing such an environment for a language would drop from the several man-years now required to the few man-days (estimated) to interface to a PI. Additionally, the existence of a common programming environment for many different languages would justify the inclusion of further capabilities.

This common programming environment provided by a PI should include facilities for: creating, modifying, storing, and retrieving programs; on-line debugging, including trace and break facilities as well as the facilities of the language for evaluation of expressions at breaks; modifying the interface between routines (via an ADVISE[6] capability); automatic spelling correction; remembering, modifying, and reissuing previous inputs; and undoing the effects of any of these PI facilities.

Such a PI has been constructed and interfaced to the programming language ECL.[7] The remainder of this paper explains the PI concept in terms of this implemented program. The deficiencies of this particular implementation are discussed in the conclusion.

SYSTEM ARCHITECTURE

The facilities provided by the implemented Programmer's Interface (PI-1) are based on the INTERLISP (formerly BBN-LISP[2]) system. In fact, they are the facilities of this system, as modified for language independence. The Programmer's Interface itself is implemented in INTERLISP and coexists with the facilities it invokes to provide the programming environment. INTERLISP was chosen as the basis both because it already had an extensive set of programming tools in an accessible form, and because their structure and operation could easily be altered to operate as required for a PI.

The system structure is shown in Figure 1. The ARPA Network[8] is used as the communications mechanism between PI-1 and the user's language processor. This choice has three advantages. First, it allows the interfacing of PI-1 to any language processor available on the ARPANET independent of what machine it runs on. Second, this interfacing can be done by PI-1 without the knowledge of the language processor. Thus no modifications to the language processor are required. Finally, the use of the Network greatly simplifies implementing the interconnection by allowing external character strings to be used for communication, rather than internal data structures with the attendant incompatibility problems.

Three properties are required of a language processor for its use with a PI:

1. There is a way to form a coroutine[9] linkage between the language processor and the PI by interconnecting their I/O ports. This type of linkage is discussed in

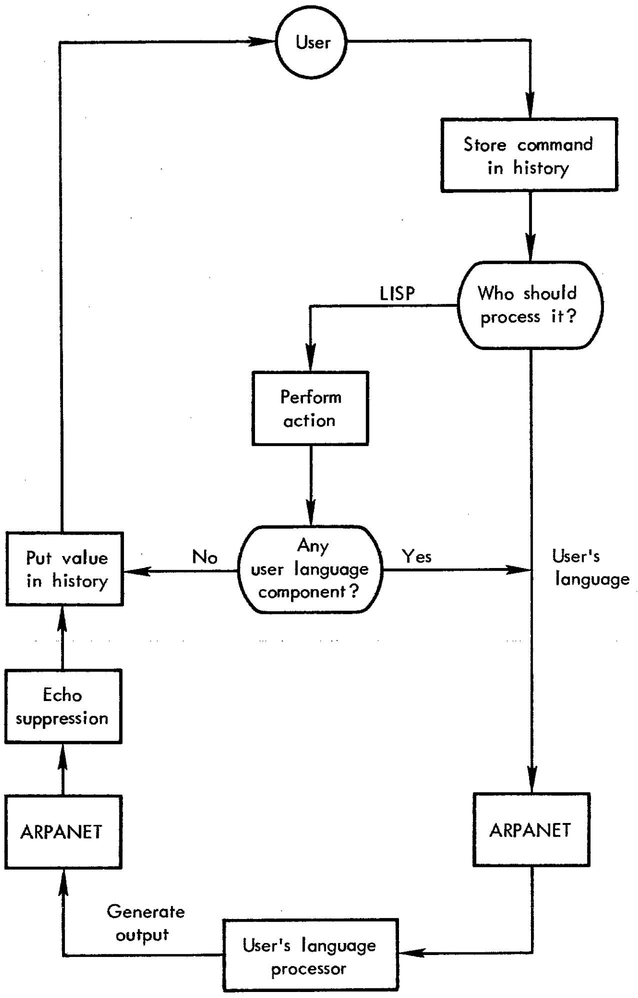


Figure 1

detail in Reference 10. With PI-1, the ARPA Network provides this linkage. Thus, for PI-1, any language processor available on the ARPANET satisfies the first requirement.

2. It has an on-line evaluator (either an interpreter or fast compiler) and can field breaks or errors within a computation.
3. It can evaluate arbitrary forms in that language either in breaks or at the top level.

PI-1 begins processing user input by storing it in a history list used by the Programmer's Assistant,[6] an INTERLISP subsystem, to retrieve, edit, group, reissue, or undo previous commands. PI-1 then examines the input to determine whether it should be processed by an INTERLISP facility or by the user's language processor. Basically, environment-type activities, such as loading files, editing programs, advising a function, etc., are performed within PI-1, while expressions in the user's language to be evaluated are passed to the language processor.

If the user's input is intended for his language processor, it is passed across the ARPA Network to that language processor. Any output generated by the processor is received across the Network again by PI-1. It suppresses the echo of the input and passes the output to the user, extracting from it the "value" and putting it into the history list for use by the Programmer's Assistant.

If the user's input is an environment-type command and should be performed within PI-1, the appropriate facility is invoked. In simple cases the operation completes, returns a value that is put in the history, and another input is processed. In more complex situations, some interaction is required with the user's language processor. This is accomplished by dynamically generating a series of inputs for the language processor that will have the desired effect or return the desired information. These are passed through the communications mechanisms to the processor; its output is captured; and either the success of the modifications is verified or the desired information is extracted. Any number of such cycles may be required before the PI-1 facility completes its processing of the user's command. As an example, considered the loading of a file. As the function definitions are read in, they are stored as a property of the corresponding atoms to be used by the PI-1's editor for any modifications required later. The function definitions also are passed to the language processor so that it can use these for evaluation. Thus, one cycle is required for each function defined in the file.

PI-1 maintains a copy of all functions defined by the user and this is used by PI-1's editor when the user alters the definition. Whenever this definition changes (by redefinition or through exiting the editor), the resulting definition is passed to the language processor as a new definition of the function.

INTERFACING A LANGUAGE TO A PROGRAMMER'S INTERFACE

Most of PI-1 is language independent, but certain portions must be modified to accept a new language. These fall into the categories of syntax modification, synchronization, program writing, and debugging.

The INTERLISP editor used by PI-1 is structural rather than string-oriented. To be effective, the text it is manipulating must have a structural basis. The syntax modification routines are responsible for introducing the structure into the user's language (only for use within PI-1). This structure is of two forms. First is the grouping of characters into lexical units. The user's language may have very different lexical grouping rules than LISP and the syntax modification package is responsible for the lexical analysis. Second, the lexical units thus produced are grouped into larger units by the use of parentheses. These units can be nested within one another to form the familiar LISP S-expression structure. The designer of the syntax modifier must decide where to introduce this structural grouping. In ALGOL-like languages, a natural place would be to group the lexical units of a statement together and groups of statements within blocks together. The structural groupings selected are introduced into all program text input by the user, and used by him to direct the editor in its modifications of this text. When this text is passed to the language processor, those structural groupings artificially introduced for editing purposes are removed before transmission.

PI-1 and the language processor must be synchronized and kept in step with each other. Logically this is very simple and is accomplished by having PI-1 wait until the language processor has completed evaluating the previous input before giving it another. This situation is signaled by the language processor's attempt to read the next input. Unfortunately (due to a deficiency in the network protocol), this information is not available. Therefore the language processor's state of readiness must be determined by examination of its output stream. Fortunately, most on-line language processors explicitly indicate their readiness for more input by providing the user with a prompt character. The language processor's output must be scanned for this prompt and this is used as a synchronization mechanism between PI-1 and the language processor.

Several facilities within PI-1, such as break, trace, and advise, cause additional statements to be written into the user's program for evaluation at runtime. The interfacer of a new language must specify the form of these additions.

PI-1 contains many advanced debugging capabilities not found in most language processors. These aids are all based on information gathered during execution or at a break point within the program. To use these facilities, the designer of the language interface must supply routines that provide the basic information on which these debugging aids are built.

PI-1 took approximately three weeks to implement and debug, including the language interface to ECL. Although no other language interfaces have yet been built, it is estimated that an interface to another suitable language could be designed, implemented, and debugged in less than a week.

PI-1/ECL EXAMPLE

The following actual example indicates the use of PI-1 with the programming language ECL. The prompt character (as defined by ECL) is either →, *, or a number followed by :>. Commentary is enclosed in square brackets.

```
→ 3+4
7
```

[Input of expression to be evaluated.]
[Answer returned.]

```
→ TEST1←EXPR(A:INT,B:INT,INT)BEGIN A+B; END;
```

[Define a function, TEST1, which takes two integer arguments A and B and returns their sum. Syntax is precisely as defined for ECL.]

```
(TEST1)
→ TEST1(3,4)
7
→ EDITF(TST1)
= TEST1
EDIT
```

[TEST1 defined.]
[Invoke TEST1 with arguments 3 and 4.]
[Answer returned.]
[Edit TEST1. Notice misspelling corrected by system.]

```
*PP

    (EXPR (A : INT, B : INT ; INT)
          (BEGIN (A + B)
                 END))
```

[Prettyprint it. Notice how structure has been added to its internal representation.]

```
*F BEGIN P
(BEGIN (A + B) END)
```

[Find the item 'BEGIN' and print what is found.]

```
*(2 (A GT B ⇒ A−B; A+B))
```

[Replace the second element, the list A+B, by the remainder of the input. This is a conditional form in ECL which evaluates A−B if A is greater than B and A+B otherwise.]

```
*PP
    (BEGIN (A GT B ⇒ (A − B))
           (A + B)
           END)
```

[Prettyprint result. Again notice how structure has been added.]

```
*(−4 (A=B → B ← 2*A))
```

[Insert rest of input before fourth element of current structure (the END item). Addition says to set B to 2*A if A=B.]

```
*PP                                      [Prettyprint it.]
    (BEGIN (A GT B ⇒ (A − B)
            (A + B)
            (A = B → (B ← 2 * A))
            END)
*UNDO                                    [User notices his error (addition made at wrong spot) and asks system
                                         to undo last command.]
(−4 − −) UNDONE.
*PP                                      [Check to see that it's really gone.]
    (BEGIN (A GT B ⇒ (A − B))
            (A + B)
            END)
*USE −3 FOR −4                           [Substitute −3 for −4 in the insertion command and reissue it.]
*PP                                      [Make sure addition put in correct spot.]
    (BEGIN (A GT B ⇒ (A − B))
            (A = B → (B ← 2 * A))
            (A + B)
            (END)
*OK                                      [Exit editor.]
TEST1
→ TEST1(3,4)                             [Test function. A is less than B, just add them.]
7
→ TEST1(4,3)                             [A greater than B, subtract B from A.]
1
→ TEST1(4,4)                             [A=B, double B and add in A.]
12
→ADVISE(TEST1 BEFORE (A← 2*A))
                                         [Modify TEST1 so that before it is entered, but after its parameters
                                         have been bound, the value of A is doubled.]
TEST1
→ TEST1(3,4)                             [Invoke modified function.]
2                                        [Double 3 to get 6 and subtract 4.]
→ USE 6 3 10 FOR 4                       [Successively substitute 6,3, and 10 for 4 in the last statement.]

18                                       [TEST1(3,6)]
3                                        [TEST1(3,3)]
16                                       [TEST1(3,10)]

→ADVISE(TEST1 AFTER (VALUE\\ ← VALUE−1))

                                         [Modify TEST1 so that after it is finished, but before it returns, the
                                         value to be returned is decremented by 1.]
TEST1
→ REDO USE                               [Reissue the previous USE command (which generated the 3 invoca-
                                         tions of TEST1)]
TYPE FAULT
− BROKEN
NIL
TYPE FAULT
− BROKEN
NIL
TYPE FAULT                               [3 type fault error occur.]
− BROKEN
NIL
3:>RETBRK(0)                             [Go back to top level.]
NIL
→ TEST1(3,4)                             [Try simple case.]
```

```
TYPE FAULT                    [Error still occurs.]
- BROKEN
NIL
1:> IN?                       [Where did error occur?]
IN ENTRY OR EXIT OF-
IN TEST1...
VALUE\\ ← VALUE - 1           [Error occurred in entry or exit of minus routine which was invoked
                              from TEST1 in the statement VALUE\\←VALUE−1. User spots
                              error (use of the undeclared variable VALUE instead of VALUE\\).]

1:>EDITF(TEST1)               [Edit TEST1.]
EDIT
*F VALUE 0 P                  [Find the use of VALUE. Go up one structured level and print
                              group.]
(VALUE\\ ← VALUE - 1)
*R VALUE VALUE\\              [Replace VALUE by VALUE\\.
*OK                           [Exit editor.]
TEST1
1:>TEST1(3,4)                 [Try test case again.]
(1)                           [Double 3, subtract 4, then decrement by 1.]
1:>                           [Go up one level of error, in this case to top level.]
NIL
→ REDO USE                    [Reinvoke previous USE command.]
(17)                          [TEST1(3,6)]
(2)                           [TEST1(3,3)]
(15)                          [TEST1(3,10)]
```

CONCLUSION

An extensive programming environment has been created for the ECL language through a program (PI-1) which allows the use of the already existing INTERLISP facilities. This greatly expands the user's facilities for creating, editing, and debugging his programs. His programming language has been transformed into a programming system. The availability of a comprehensive set of "environment" tools working in conjunction with the programmer's language is extremely important to his productivity.

The significance of this work, however, lies not in the particular interface provided between INTERLISP and ECL, nor in the extensive capabilities provided the user, but rather, in (1) the observation that very little of the interface itself, or of the capabilities provided, are language dependent, (2) the recognition that the programming environment can be effectively split into an "environment" part and an execution and evaluation part and (3) the experience gained from building such a system and interfacing a language to it.

PI-1, however, suffers from a number of deficiencies, the most important of which is the use of already existing tools in more general environments than they were designed for. This was most notable in the use of LISP's editor for non-structured text (and the need therefore to introduce structure by parentheses) and the need to replace LISP's input routines to provide the proper lexical analysis for the interfaced language. Both of these problems could be avoided in a PI by having it use the syntax description of the language to guide the input, and editing and display of programs.

One of the strengths of the PI concept is the split between the "environment" part and the evaluation part. This split, however, introduces the problem of communication and synchronization; each part must keep the other informed about changes it makes that affect the other. In PI-1, this communication and synchronization was partial and clumsy. The flow of information from the environment to the evaluation part was adequate, but the reverse flow was not. The need to communicate to another program suitable explanations of what the state of the evaluation was, what the cause of the error was, or even that an error occurred was simply not envisioned or planned for.

PI-1 has thus demonstrated that a moderately integrated PI can be built that has facilities for beyond what is typically available at a fraction of the cost. However, development of highly integrated PI will have to await a better understanding of the functional requirements of a language processor in such an environment.

Although the Programmer's Interface has only been interfaced to one language (ECL), and although it only contains a small fraction of the capabilities ultimately desired, it is having a major effect by acting as a prototype for a major software project[11,12] being undertaken to develop this understanding and provide a single, common, comprehensive programming environment interfaced to a wide variety of languages running on many different machines communicating through a network. New languages or machines could be interfaced to the system at a fraction of the cost of providing a separate programming environment. Widespread usage would justify the expenditure of more resources to augment and improve the capabilities provided. Such a PI could free users from having to develop their programs only with software available on their own machines and could provide a much more comprehensive and coordinated software development package than is currently available.

REFERENCES

1. Rulifson, J. F., J. A. Derksen, and R. J. Waldinger, *QA4: A Procedural Calculus for Intuitive Reasoning*, Stanford Research Institute Artificial Intelligence Center, Technical Note 73, November 1972.
2. Teitelman, W. D., G. Bobrow, A. K. Hartley, and D. L. Murphy, *BBN-LISP TENEX Reference Manual*, BBN Report, July 1971.
3. Falkoff, A. D., and K. E. Iverson, *The APL Terminal System: Instructions for Operation*, IBM Corporation, T. J. Watson Research Center, Yorktown Heights, New York, March 1967.
4. Kemeny, J. G., and T. E. Kurtz, *BASIC Programming*, John Wiley and Sons, Inc., New York, 1967.
5. IBM Corporation, *O/S Time Sharing Option: PL/I Checkout Compiler*, Form SC33-0033, November 1971.
6. Teitelman, W., "Automated Programming—The Programmer's Assistant," *AFIPS Conference Proceedings*, 1972, Vol. 41, Part II, pp. 917-921.
7. Wegbreit, B., "The ECL Programming Systems," *AFIPS Conference Proceedings*, 1971, Vol. 39, pp. 253-262.
8. Roberts, L. G., and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Conference Proceedings*, 1970, Vol. 36, pp. 543-549.
9. Conway, M., "Design of a Separable Transition-Diagram Compiler," *Communications of the ACM*, Vol. 6, No. 7, July 1963, pp. 396-398.
10. Balzer, R. M., *Ports—A Method for Dynamic Interprogram Communication and Job Control*, The RAND Corporation, August 1971.
11. Balzer, R. M., T. E. Cheatham, S. D. Crocker, and S. Warshall, *National Software Works Design*, USC/Information Sciences Institute, RR-73-16, November 1973.
12. Balzer, R. M., T. E. Cheatham, S. D. Crocker, and S. Warshall, *The National Software Works*, USC/Information Sciences Institute, RR-XX-18, December 1973.

Guidelines for the use of infinite source queueing models in the analysis of computer system performance*

by J. P. BUZEN
Harvard University
Cambridge, Massachusetts
and
Honeywell Information Systems
Waltham, Massachusetts
and
P. S. GOLDBERG
Case Western Reserve University
Cleveland, Ohio

INTRODUCTION

Mathematical models based on queueing theory are widely used in the analysis of computer system performance.[3] As in the case of other engineering disciplines, these models never correspond exactly to the real systems they are intended to represent. However, if the associated error terms are sufficiently small the models can still serve as valuable tools for estimating performance levels in specific cases and for studying the factors which influence overall system behavior. In this paper we examine some error terms which arise when the familiar M/G/1 queueing model[2] is used to predict expected response times and queue lengths in systems which contain only a finite number of sources.

To illustrate the essence of the problem suppose it is necessary to determine the mean response time (i.e., waiting time+service time) for a particular component or subsystem that is being considered for use within a larger system. Queueing theory provides a set of formulas for estimating this quantity once the service time distribution and the arrival process are known. Figure 1 illustrates the M/G/1 queueing model which is commonly used in such cases. To apply this model it is necessary to determine the mean rate at which requests will arrive at the subsystem and also the distribution of the subsystem's service time. The M/G/1 model then assumes the following two conditions are true:

A. Service times are given by independent identically distributed random variables having the empirically observed distribution.
B. Requests arrive according to a Poisson process which has the empirically observed arrival rate.

Given A and B it is possible to demonstrate that the expected response time of the subsystem is[1]

$$R=\frac{1}{\mu}+\frac{\rho(1+c)}{2\mu(1-\rho)} \tag{1}$$

where

μ = the reciprocal of the mean service time
ρ = the mean service time to mean inter-arrival time ratio (the load on the subsystem)
c = the variance to mean squared ratio for the service time distribution

FINITE AND INFINITE SOURCE MODELS

Although equation 1 is relatively easy to apply, it is only of interest if it yields values of R which are reasonably close to the response times one would actually observe in real systems. This depends in part on the extent to which assumptions A and B are satisfied in real systems. Assumption A should be reasonably accurate in many cases since individual service times are typically generated by independent processes and thus tend to be serially uncorrelated. However, assumption B is often suspect because of the finite source nature of the arrival process.

This point is illustrated in Figure 2. Assume that this figure presents an exact description of some existing system. The system contains N customers (programs), each circulating counterclockwise through the loop and undergoing successive periods of independent activity, queueing delay and subsystem service. In the case where service times are given by arbitrarily distributed random variables and independent activity times by exponentially distributed random variables, the system depicted in Figure 2 is known as a finite source M/G/1/N queueing system.

One of the significant aspects of the M/G/1/N model is that the rate of arrival of new customers to the subsystem steadily decreases as queue length increases, and in fact

* This work was sponsored in part by the Electronic Systems Division, U.S. Air Force, Hanscom Field, Bedford, Massachusetts under Contract Number F19628-70-C-0217.

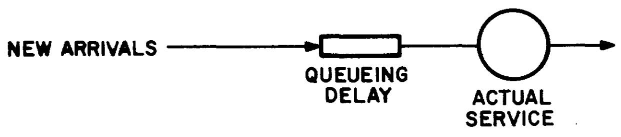


Figure 1—Infinite source model

drops to zero when all N customers are waiting for or receiving service. However, the arrival rate will not vary greatly if N is fairly large and queue length remains relatively short most of the time. In such cases it is reasonable to assume, as an approximation, that the mean arrival rate remains constant regardless of the actual length of the queue.

This is precisely the approximation that is made in the M/G/1 model. It is further assumed in this model that customer service times are given by arbitrarily distributed random variables and that inter-arrival times are exponentially distributed. Thus the M/G/1 model can be regarded as the limiting case of the M/G/1/N model in which N approaches infinity. Of course it is necessary to assume that mean independent activity time also approaches infinity as part of the limiting process in order to prevent the subsystem from becoming overloaded as N grows large.

INFINITE SOURCE ADVANTAGES

Although it is possible to explicitly solve the "finite source" M/G/1/N model and derive formulas for quantities such as expected response time (see equation 2 in Appendix), there are several reasons why the "infinite source" M/G/1 model is often preferred. The primary reason is mathematical simplicity. For example, in the basic case of first come first served scheduling, the formula for expected response time is significantly easier to evaluate in the M/G/1 case (equation 1) than it is in the M/G/1/N case (equation 2). This greater complexity is present in almost all finite source formulas.

Another consequence of the relative mathematical simplicity of the M/G/1 model is the fact that infinite source solutions exist for certain problems which have not yet been solved in the finite source case. Schrage's analysis of the infinite level foreground/background scheduling discipline is a case in point.[4]

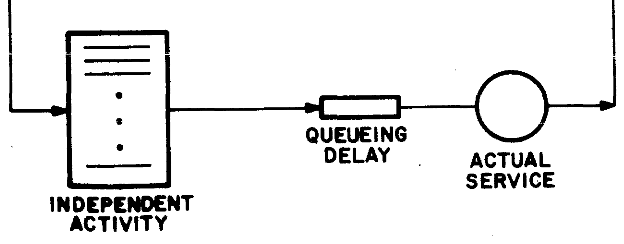


Figure 2—Finite source model

A somewhat different reason for preferring the infinite source model is that it is sometimes easier to measure the mean arrival rate at a subsystem (a local measurement) than it is to measure the independent activity times at remote sites (e.g., user terminals in a time-sharing system). A final advantage is derived from the fact that the parameter N does not explicitly enter into the infinite source model. Thus the exact value of N need not be considered when applying the model to a specific system. This is particularly useful in cases where the actual value of N fluctuates as a result of external factors.

INFINITE SOURCE ERROR TERMS

Given the advantages of infinite source models, it is natural to consider the question of how accurate these models are in predicting performance levels of real systems which, in fact, almost always contain only a finite number of sources. To study this question, suppose that an actual system which conforms exactly to the M/G/1/N description is being analyzed with the aid of an infinite source M/G/1 model. Since both models can be solved explicitly it is possible to compute both the "exact" mean response time R_N (equation 2) and the "approximate" mean response time R (equation 1). Assuming the M/G/1 model has been properly calibrated to the M/G/1/N system so that the service time distributions and mean arrival rates are identical, it is then possible to obtain the error term $(R-R_N)/R_N$. This error is directly attributable to the infinite source assumption. As an additional point, it can be shown that the relative error in expected response time is identical to

TABLE I—Relative Errors for Constant Service Times

	ρ								
N	.1	.2	.3	.4	.5	.6	.7	.8	.9
1	.059	.127	.215	.332	.508	.742	1.172	2.031	4.375
2	.030	.068	.117	.188	.296	.456	.725	1.280	2.752
3	.020	.046	.082	.135	.213	.344	.541	.995	2.239
4	.014	.036	.066	.105	.171	.276	.455	.819	1.865
5	.011	.028	.053	.090	.142	.228	.382	.684	1.590
6	.010	.024	.045	.076	.124	.199	.329	.597	1.480
7	.008	.021	.038	.065	.108	.173	.297	.569	1.293
8	.007	.018	.034	.059	.095	.162	.267	.502	1.226
9	.006	.016	.030	.051	.088	.141	.248	.466	1.132
10	.006	.015	.028	.047	.078	.133	.230	.430	1.021
20	.003	.008	.014	.025	.042	.074	.134	.262	.716
30	.002	.005	.009	.017	.028	.052	.094	.202	.587
40	.001	.004	.007	.013	.022	.039	.073	.156	.436
50	.001	.003	.006	.010	.018	.032	.062	.139	.392
60	.001	.003	.005	.008	.015	.026	.054	.116	.349
70	.001	.002	.004	.007	.013	.023	.045	.098	.339
80	.000	.002	.003	.006	.011	.020	.040	.090	.297
90	.000	.001	.003	.006	.010	.018	.036	.083	.258
100	.000	.001	.003	.005	.009	.016	.033	.077	.250
120	.000	.001	.002	.004	.008	.014	.028	.068	.235
140	.000	.000	.002	.003	.006	.012	.023	.055	.196
160	.000	.000	.002	.003	.005	.010	.020	.050	.185
180	.000	.000	.001	.002	.005	.009	.018	.045	.153
200	.000	.000	.001	.002	.004	.008	.016	.042	.145

TABLE II—Relative Errors for Erlang-2 Service Times

	ρ								
N	.1	.2	.3	.4	.5	.6	.7	.8	.9
1	.088	.190	.322	.498	.762	1.113	1.758	3.047	6.563
2	.044	.102	.180	.290	.442	.674	1.061	1.825	4.012
3	.031	.070	.127	.204	.321	.495	.789	1.345	3.014
4	.021	.054	.098	.160	.253	.402	.650	1.140	2.591
5	.017	.045	.079	.133	.213	.336	.556	.946	2.217
6	.014	.036	.068	.113	.180	.296	.487	.886	1.978
7	.012	.031	.057	.097	.164	.260	.421	.772	1.812
8	.011	.028	.051	.088	.140	.229	.383	.735	1.590
9	.010	.025	.046	.080	.129	.216	.358	.653	1.505
10	.009	.022	.042	.071	.121	.197	.334	.610	1.398
20	.004	.012	.021	.037	.063	.109	.196	.372	.945
30	.003	.008	.015	.026	.045	.078	.142	.298	.732
40	.002	.006	.011	.020	.033	.059	.112	.238	.657
50	.002	.005	.009	.015	.027	.049	.091	.189	.571
60	.002	.004	.007	.013	.023	.042	.080	.172	.483
70	.001	.003	.006	.011	.020	.035	.071	.148	.436
80	.001	.003	.006	.010	.018	.031	.060	.137	.390
90	.001	.003	.005	.009	.016	.028	.055	.127	.381
100	.001	.002	.005	.008	.014	.025	.050	.110	.338
120	.001	.002	.004	.007	.012	.021	.043	.097	.290
140	.001	.002	.003	.006	.010	.019	.038	.087	.275
160	.001	.001	.003	.005	.009	.016	.031	.080	.263
180	.001	.001	.003	.004	.008	.015	.028	.067	.224
200	.000	.001	.002	.004	.007	.013	.026	.062	.214

the relative error in expected queue length. Thus the value of $(R-R_N)/R_N$ can be interpreted in either manner.

Tables I through IV present values of $(R-R_N)/R_N$ for a wide range of model parameters. The service time distributions used in the four tables are constant, Erlang-2,

TABLE III—Relative Errors for Exponential Service Times

	ρ								
N	.1	.2	.3	.4	.5	.6	.7	.8	.9
1	.117	.254	.430	.664	1.016	1.484	2.344	4.063	8.750
2	.059	.135	.237	.379	.588	.894	1.348	2.387	5.339
3	.041	.093	.168	.274	.411	.641	1.007	1.764	3.794
4	.028	.073	.131	.217	.334	.524	.809	1.453	3.159
5	.023	.060	.105	.175	.274	.443	.696	1.240	2.672
6	.019	.048	.091	.149	.240	.382	.615	1.085	2.461
7	.017	.042	.077	.134	.212	.338	.566	1.006	2.145
8	.015	.037	.068	.116	.189	.308	.492	.919	1.935
9	.013	.033	.062	.106	.169	.273	.475	.829	1.860
10	.012	.030	.057	.098	.158	.259	.433	.782	1.762
20	.006	.016	.029	.050	.088	.148	.255	.473	1.157
30	.004	.011	.020	.035	.060	.102	.189	.370	.950
40	.003	.008	.015	.027	.045	.082	.151	.302	.786
50	.002	.006	.012	.022	.037	.065	.124	.260	.699
60	.002	.005	.010	.018	.031	.056	.104	.211	.608
70	.002	.005	.009	.015	.027	.049	.093	.195	.519
80	.001	.004	.008	.013	.024	.044	.084	.182	.510
90	.001	.004	.007	.012	.022	.038	.077	.159	.461
100	.001	.003	.006	.011	.020	.034	.066	.149	.415
120	.001	.003	.005	.009	.017	.029	.057	.124	.399
140	.001	.002	.004	.008	.014	.025	.050	.112	.348
160	.001	.002	.004	.007	.013	.022	.045	.102	.302
180	.001	.002	.003	.006	.011	.020	.041	.094	.290
200	.001	.002	.003	.006	.010	.018	.035	.087	.279

TABLE IV—Relative Errors for Hyperexponential Service Times

	ρ								
N	.1	.2	.3	.4	.5	.6	.7	.8	.9
1	.147	.317	.537	.830	1.270	1.856	2.930	5.078	10.938
2	.074	.170	.306	.469	.720	1.080	1.666	2.921	6.366
3	.052	.117	.210	.340	.526	.798	1.249	2.175	4.598
4	.036	.092	.164	.270	.420	.655	1.013	1.725	3.932
5	.029	.076	.139	.219	.356	.574	.880	1.483	3.443
6	.024	.060	.115	.194	.305	.472	.752	1.310	3.071
7	.021	.053	.102	.168	.262	.420	.694	1.220	2.570
8	.018	.047	.087	.146	.242	.373	.608	1.124	2.357
9	.016	.042	.079	.134	.218	.353	.558	1.025	2.289
10	.015	.038	.072	.124	.196	.315	.540	.924	2.194
20	.008	.020	.037	.065	.111	.183	.318	.581	1.394
30	.005	.014	.025	.045	.077	.134	.239	.444	1.088
40	.004	.010	.019	.035	.061	.103	.185	.367	.922
50	.003	.008	.016	.028	.048	.087	.153	.303	.835
60	.003	.007	.013	.023	.041	.072	.135	.280	.739
70	.002	.006	.011	.020	.035	.063	.115	.246	.643
80	.002	.005	.010	.017	.031	.056	.104	.216	.592
90	.002	.005	.009	.016	.028	.051	.096	.202	.542
100	.002	.004	.008	.014	.025	.044	.088	.191	.533
120	.001	.003	.007	.012	.021	.038	.072	.160	.437
140	.001	.003	.006	.010	.018	.033	.064	.135	.422
160	.001	.002	.005	.009	.016	.029	.057	.124	.371
180	.001	.002	.004	.008	.015	.026	.052	.115	.325
200	.001	.002	.004	.007	.013	.024	.047	.106	.313

exponential and hyperexponential respectively. Thus the tables present distributions whose variance to mean squared ratios increase progressively through the range: 0, ½, 1, ⅗.

The other parameters in the tables are the number of sources in the actual finite source system (N) and the empirically observed load (ρ) placed on the subsystem. Conceptually, ρ is first measured for the M/G/1/N system and then automatically duplicated in the M/G/1 model when the input rate and service time distribution are assigned. Note that ρ is also the subsystem utilization factor (i.e., the proportion of time the subsystem is busy). For a more detailed description of the parameters involved and the manner in which the tables were constructed, see the Appendix.

DISCUSSION

The description of finite and infinite source models presented earlier suggests that the M/G/1 model should be a relatively good approximation to an M/G/1/N system when N is large. Although the tables support this general observation, they also point out the sensitivity of the error factors to the value of ρ. That is, the value of N necessary to attain a given error level is much smaller for low and moderate values of ρ than it is for values of ρ which are close to one. Thus the infinite source assumption is of questionable value when ρ is large.

This observation has a number of implications for infinite source models in general. When such models are used to evaluate the relative differences between particular scheduling disciplines or the sensitivity of individual scheduling

TABLE V—Minimum Number of Sources Necessary to Attain a 5 Percent Error Level

Distribution	.1	.2	.3	ρ .4	.5	.6	.7	.8
Constant	2	3	6	10	17	32	61	160
Erlang-2	2	5	9	16	27	49	101	200+
Exponential	3	6	12	21	38	70	142	200+
Hyperexponential	4	8	16	27	48	93	188	200+

disciplines to parameters such as quantum size, one usually finds that the greatest differences are associated with the larger values of ρ. However, these are precisely the values of ρ for which infinite source models are least accurate and therefore most misleading. Thus, even though infinite source models may be acceptable in a variety of engineering applications that involve small- or mid-range values of ρ, the large error factors that arise as ρ approaches one may seriously undermine the value of these models in many theoretical contexts.

Another point which the tables illustrate is that the value of N necessary to attain a given error level increases as the variance to mean squared ratio increases. This effect is explicitly illustrated in Table V which presents the minimum value of N necessary to attain a 5 percent error level as a function of the variance to mean squared ratio and also the value of ρ. Similar tables can be constructed for any other error levels of interest by using the data in Tables I through IV. In addition, error values for any service time distribution with a variance to mean squared ratio between 0 and $\frac{3}{2}$ can be estimated from the tables by interpolation.

REFERENCES

1. Cox, D. R. and W. L. Smith, *Queues*, Methuen, London, 1961.
2. Kendall, D. G., "Some problems in the theory of queues," *J. Roy. Stat. Soc. B*, 13, 2 (1951), pp. 151-185.
3. McKinney, J. M., "A survey of analytic time sharing models," *Computing Surveys*, 1, 2 (1969), pp. 105-116
4. Schrage, L. E., "The queue M/G/1 with feedback to lower priority queues," *Management Science*, 13, 2 (1967), pp. 466-474.
5. Takacs, L., "On a stochastic process concerning some waiting time problems," *Theory of Probability and its Applications*, 2, 1 (1957), pp. 90-103.

APPENDIX

The expected response time for a finite source M/G/1/N queueing model is given by[5]

$$R_N=\frac{N}{\mu}-\frac{1}{\lambda}+\left[\lambda\sum_{r=0}^{N-1}\binom{N-r}{r}\frac{1}{H_r}\right]^{-1} \qquad (2)$$

where

μ = the reciprocal of the mean service time
λ = the reciprocal of the mean independent activity time
N = the number of sources

$$H_r=\begin{cases}1 & \text{for } r=0\\ \prod_{i=1}^{r}\varphi(i\lambda)/(1-\varphi(i\lambda)) & \text{for } r=1,2,\ldots,N-1\end{cases}$$

$$\varphi(s)=\int_0^\infty e^{-st}dF(t)$$

$F(t)$ = the cumulative service time distribution

In order to compute the relative error values $(R-R_N)/R_N$ which appear in Tables 1-4 it is necessary to specify the following quantities: the service time distribution, the load on the server (ρ) and, for the M/G/1/N case, the number of sources (N). The service time distributions used in the four tables are as follows:

Table I.	Constant service time, mean = 1, variance to mean squared ratio = 0.
Table II.	Erlang-2 service time, distribution = $4te^{-2t}$ (sum of two exponentials each having a mean of $\frac{1}{2}$), mean = 1, variance to mean squared ratio = $\frac{1}{2}$.
Table III.	Exponential service time, distribution = e^{-t}, mean = 1, variance to mean squared ratio = 1.
Table IV.	Hyperexponential service time, distribution = $e^{-2t}+\frac{1}{3}e^{-2t/3}$ (equal mixture of exponentials with means of $\frac{1}{2}$ and $\frac{3}{2}$), mean = 1, variance to mean squared ratio = $\frac{3}{2}$.

Once ρ is specified, computation of R by use of equation 1 is entirely straightforward. The major difficulty arises in the computation of R_N. This is because equation 2 does not depend explicitly on the value of ρ but rather on N, λ and the cumulative service time distribution $F(t)$. Hence it is necessary to compute λ from ρ, N and $F(t)$ before applying equation 2.

In principle, λ can be computed from ρ, N and $F(t)$ by using equation 3.

$$\rho=\lambda N\left[\lambda N+\sum_{r=0}^{N-1}\binom{N-r}{r}\frac{1}{H_r}\right]^{-1} \qquad (3)$$

However, the complex dependency of H_r on λ makes an exact solution for λ impossible. Thus the basic strategy used in constructing the tables is to first obtain an approximate solution to equation 3 by numerically determining a value λ' which satisfies equation 4.

$$\left|\rho-\lambda' N\left[\lambda' N+\sum_{r=0}^{N-1}\binom{N-r}{r}\frac{1}{H_r}\right]^{-1}\right|<.005 \qquad (4)$$

This value of λ' is then used in equation 2 to compute R_N, and the corresponding value of R is computed from equation 1. The value of ρ used in the computation of R is obtained by substituting λ' into equation 3. Thus the finite and infinite source models will have the identical value of ρ, and this value will always be within .005 of the corresponding column headings which appear in the tables.

Data base concepts applied to generalized programming packages

by G. CORT STEINHORST and BARRY L. BATEMAN

Texas Tech University
Lubbock, Texas

and

DANIEL L. CURTIS

University of Southwestern Louisiana
Lafayette, Indiana

INTRODUCTION

Software "packages" are becoming more and more prevalent in both use and availability. These packages provide many useful functions, without the costly, time-consuming development and testing normally required to implement the desired operations. A disadvantage of many packages, however, is the difficulty in modifying them in order to receive the desired result, if the original resultant of the package is not totally satisfactory.

TYPICAL TECHNIQUES

Various approaches have been used in package implementation. A common type of package is a series of subroutines (e.g., the Scientific Subroutine Package provided by most major computer manufacturers). This type of package provides great assistance to the experienced programmer but is of little use to the non-computer-oriented researcher. The experienced user can easily modify these routines to meet his particular qualifications. The inexperienced user must normally "make do" with his output, must seek technical assistance, or, in extreme cases, must forgo the use of the computer in his research.

Another type of package consists of a series of separate programs (e.g., The Biomedical Programs). These programs are generally very flexible. A disadvantage is that many times several of these programs must be run in sequence in order to obtain the desired results. Normally, this requires many similar calculations, which are repeated with each execution of the separate programs. There is little possibility of executing the entire set of calculations in one step. The usual procedure is to execute one program, check the results, execute the next program, check the results, and so on until the final answers are obtained. This process requires a relatively large block of time and also requires understanding the function of several programs in order to achieve the final results.

A BETTER APPROACH

For the average researcher, with little computer knowledge, packages can be written which employ techniques developed in data base management systems. The main advantages to this approach are (1) a simple, user-oriented "language" can be developed to direct the processes; (2) all input data and intermediate calculations can be stored in a common, "data base" area; (3) modular programming can be used to allow easy modification of the package without extensive reprogramming.

The user "language" need not be a very sophisticated one. The basic purpose of this language is to provide easy direction of processes by non-computer-oriented researchers. A very likely possibility for the language would be the use of the names of the particular calculations desired, followed by the data, in free form, which is to be used in the calculations. By storing the input data and the intermediate calculations in an area common to all routines, it is possible to perform a complex series of operations upon the data with little redundant calculation. In outline form this type of package consists of (1) an executive, language-interpreter module, (2) a common data base, and (3) a series of operation modules and sub-modules, which operate on the data base.

The executive program determines the sequence of operations and directs the order of calls to each of the operation routines (modules). The data base provides an easy method of determining if an intermediate operation has been performed and also provides a common area for communication among the various modules and sub-modules of the package.

APPLICATION OF DATA BASE CONCEPTS

After considering the problems related to the so-called "packages," an attempt has been made to design an improved package. An example of this technique is STATPAK (Statistical Package) and was implemented on the RCA Spectra 70/45.[1] STATPAK is designed with the intentions

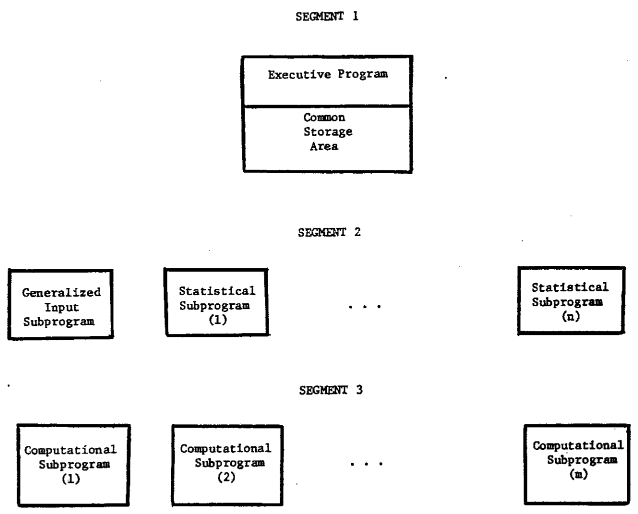

SEGMENT 1
Executive Program
Common Storage Area
SEGMENT 2
Generalized Input Subprogram
Statistical Subprogram (1)
. . .
Statistical Subprogram (n)
SEGMENT 3
Computational Subprogram (1)
Computational Subprogram (2)
. . .
Computational Subprogram (m)

Figure 1—STATPAK systems

of: (1) Providing the user a package which does not require the knowledge of programming; (2) Eliminating duplication of input data; and (3) Providing a common area to store information to eliminate duplication of calculations.

The STATPAK system, as shown in Figure 1, is composed of three segments: (1) an executive program and a common data storage area; (2) major statistical subprograms and a generalized input subprogram; and (3) minor computational subprograms. The executive program acts as both the communicator between the user and the subprograms and the communicator among the subprograms themselves. The major statistical subprograms included in the second segment consist of the larger subprograms such as hypothesis testing, analysis of variance and regression analysis. The generalized input subprogram accepts the input data in varying format, which allows the data to be in any form. The minor computational subprograms calculate various quantities, such as sums, sums of squares, means and variances, which are required by many of the statistical subprograms.

In implementing the system a series of statistical subprograms were selected to be included within the package. The system was designed in such a manner that if the package, consisting of the selected subprograms, would prove to be feasible, the addition of other statistical subprograms would be a trivial task.

MINIMIZING CALCULATIONS

A technique was devised to eliminate repeating a calculation by checking indicators to determine if a given statistical subprogram has already been executed. This technique is accomplished through the use of an array which is located in the common storage area. This array contains values of one and zero which indicate that either a statistical subprogram has been executed or has not been executed. Each statistical subprogram is assigned a number, i, and upon completion of the execution of the ith statistical subprogram, a "1" is placed in the ith element of the array. Therefore, any statistical subprogram has the capability of determining if any statistical subprogram, including itself, has been executed.

When writing each statistical subprogram, the results of every calculation which could possibly be required by another statistical subprogram are placed in the common storage area. By placing the results of calculations in an area accessible to all statistical subprograms, calculations are not repeated. For example, if the mean is required in a statistical subprogram, the indicators for the statistical subprograms, which calculate or require the mean, are checked. If any of these statistical subprograms have been executed, the mean is already available in the common storage area, and thus, a calculation is not necessary. If none of the statistical subprograms which use the mean have been executed, control is transferred to the computational subprogram which calculates the mean. Once the mean has been calculated, control is returned to the calling statistical subprogram. Prior to halting that statistical subprogram, a "1" is placed in the indicator which references this particular subprogram; therefore, the mean will not be calculated again.

GENERALIZED INPUT

Another subprogram in the second segment of the STATPAK system is the generalized input subprogram. This subprogram allows free format of data. In other words, the user need not have his input data in a particular form. Another feature of the input subprogram is that there are virtually no limits placed on the amount of data points or the number of variables. Most programs or subprograms have a limit to the number of data points or the number of variables because the data is usually maintained in a rectangular matrix of fixed dimensions within the program or subprogram. In these programs or subprograms, a matrix dimensioned n by m will allow a maximum of n data points of m variables or vice versa. In STATPAK's input subprogram, a single-dimensioned array of n times m locations is used. Thus, if there are k variables, the number of allowable data points is $[n*m/k]$ where $[\ ]$ implies the largest integer such that $k*[n*m/k]\leq n*m$. For example, if an array contains 1000 locations and the number of variables is 30 then the number of allowable data points is [1000/30], which equals 33.

STORAGE OF DATA

In addition to allowing a varying number of data points or variables within the array, the input subprogram writes the array to disk once the array is filled. The input subprogram then places a "1" in an indicator in the common storage area; thus, each statistical subprogram can determine if the data are in common storage by checking an indicator. If the indicator contains a zero, the statistical subprogram retrieves the information stored on the disk. The technique of storing the data, allowing the data to be transferred to disk, and free format of data make the input requirements of the STATPAK system very flexible.

EXECUTIVE PROGRAM

To provide the user with a package which does not require a knowledge of programming, an executive program was written which uses a series of control cards and performs the functions indicated by the code on each card. In addition to providing ease of use, the executive program provides the user with other desirable features. The user may reference several statistical subprograms for one set of data; thus, eliminating the need of duplicating common input requirements of each subprogram. Also, the user may execute a specified series of statistical subprograms for any number of data sets.

The executive program reads in the control cards which constitute one job, and performs the job X times using the X sets of data that follow the END JOB card. This process is continued until a STATPAK END card is encountered, which terminates the system. The executive program also has the function of reinitializing the array of indicators of the statistical subprograms for each execution of a job or for the execution of a different job. Since each execution of a job or the execution of a different job requires a new set of data, the results stored in the common storage area are for the previous job iteration or previous execution of a different job; therefore, it is necessary for all calculations to be repeated.

The executive program has a list of the contained statistical subprograms and the corresponding codes for each. As each control card following the JOB card is interrogated, the code is matched to the desired statistical subprogram and control is transferred to that subprogram.

CONCLUSIONS

Generalized software packages can be implemented which will provide valuable assistance to the researcher. By using data base technology, it is possible to create a flexible organization which allows maximum computer usage with a minimum of technical ability. Simple semantic capability along with good error messages augment the generalized approach.

REFERENCES

1. Curtis, Daniel L., *STATPAK; A Statistical Package for the RCA 70/45*, Masters Thesis, University of Southwestern Louisiana, Lafayette, Louisiana, 1968.
2. Dixon, W. J., ed., *BMD, Biomedical Computer Programs*, Los Angeles, California, 1969.
3. RCA, *Spectra 70 Scientific Subroutine System*, Manual No. 79-35-319, March, 1968.

On-line user-computer interface—The effects of interface flexibility, terminal type, and experience on performance

by GEORGE H. WALTHER

U.S. Air Force Academy
Colorado Springs, Colorado

and

HAROLD F. O'NEIL, JR.

The University of Texas
Austin, Texas

INTRODUCTION

In less than two decades the electronic digital computer has evolved from a high-speed replacement for the abacus to a full-fledged partner in dialogue with humans. By its very nature, on-line computing thrusts the user into an entirely different environment than does conventional batch processing. The problems inherent in a person's being made a system component—in a sense an extension of the computer hardware—were largely ignored until quite recently. Since the early users of on-line systems were highly skilled professionals who were both willing and able to communicate in terms most convenient for the machine, few programmers were aware of the user-computer communications gap, and even fewer were concerned about it. However, the lowered cost of computer access and the proliferation of on-line systems produced a new breed of users, people whose expertise was in some area other than computer technology. As their initial fascination with conversational computing wore off, users reported experiencing feelings of intense frustration and of being "manipulated" by a seemingly unyielding, rigid, intolerant dialogue partner, and these users began disconnecting from time-sharing services at a rate which was very alarming to the industry.

Perhaps one of the best and most recent statements of the recognition by the computer industry that considerable attention should be given to the user comes from Martin[1] of the IBM Systems Research Institute:

> "*Increasingly . . . ,* man *must become the prime focus of system design. The computer is there to serve him, to obtain information for him and to help him do his job. The ease with which he communicates with it will determine the extent to which he uses it. Whether or not he uses it powerfully will depend upon the man-machine language available to him and how well he is able to understand it* (p. 3)."

A review of the literature[2] clearly indicates that a shift of emphasis is currently in progress—from a deep concern for the elegance of algorithms to varying degrees of interest in satisfying individual users. But how does one discover the "best" method for designing a user interface? Can the user simply be asked what he would like to have happen when he sits down at a terminal device? Apparently not, according to several recent writers on the subject. Further, quite contrary to popular opinion, "armchair" intuitive design techniques have not proved to be a sufficient basis for even the most concerned systems designers to use. The study being reported on is a case in point. It was the intuitive feeling of the present writers that interface flexibility would be uniformly "good" for all users, but the data did not support this contention.

Much opinion has been offered as to what constitutes user-oriented design practices, but very few intuitions have been supported with data. Bennett[2] called for a transformation of the current art of system design into an engineering discipline. The United States Air Force is requiring human factors analyses of software design in new systems. Some writers have suggested that a common user interface should be the goal, presumably leading to a set of conventions which, once discovered, would maximize ease-of-use perceptions and feelings of satisfaction on the part of users. Others point out that the differences which make individuals unique overshadow commonalities, so there is considerable doubt that any interface standards would please even most of the users most of the time.

One approach, and the one chosen for this study, is to allow the interface to be alterable by the user under operating conditions without the necessity for reprogramming. Therefore, the user interface is programmed with the capability for making differential responses to a variety of users under a wide range of conditions. This is what will be referred to as interface flexibility.

PROBLEM STATEMENT AND METHODOLOGY

This study investigates whether interface flexibility is a viable solution to the problem of giving the on-line interface the quality of adaptability. The question is asked whether interface flexibility, operationalized as options offered to the user of an applications program, is "good" for everyone's performance with their program, irrespective of their personal attributes. Where flexibility appears not to be universally the best approach for all users, an attempt is made to ascertain the kind of users to whom it should be offered.

The CDC 66/6400 on-line computer system of the University of Texas at Austin was used for this research. Four terminal devices were employed: two standard Model 33-KSR Teletypes, and two Datapoint 3300 cathode ray tube (CRT) keyboard terminals. Although the CRT's and the on-line system were capable of 300-baud display rates, these terminals were buffered down to teletypewriter speeds (110-baud) so that CRT users would have no terminal display performance advantage over teletypewriter users. Terminals accessed the on-line computer via acoustic data couplings to ordinary voice-grade dial-up commercial telephone lines.

All measurements, instruction, data collection and tasks were administered by the on-line computer system of programs created expressly for this experiment. Programs were written either in SNOBOL4 or FORTRAN IV and were compiled into object code for the CDC 66/6400 system. Nine independent programs were called sequentially to administer the experiment to each user. Programs were modularized to provide re-start checkpoints in the event of system malfunctions from which full recovery was not possible and to reduce core storage requirements. A journal file was maintained on disk for each user. It contained such information as user personal data, timing statistics, verbatim transcripts of all messages issued by the user, and program instrumentation data. The applications program was instrumented to record user response time and system response time, syntax errors made by the user, counters of the frequency of use of flexibility options, and a copy of the target text file as it appeared following being edited by the user.

Sixty-nine undergraduate computer science students, all of whom had successfully completed at least the basic FORTRAN programming course and were nearing the end of either the intermediate programming course or a more advanced Survey of Programming Languages course, served as users in this study. This was the first encounter with an on-line system for about 41 percent of them. Another 38 percent claimed experience levels ranging from "once or twice before" to "several times before" (they will be referred to as users with "*some* experience" in this report), and 22 percent reported having "much experience."

An on-line text editor served as the experimental vehicle for the study. It was specifically designed and constructed for this experiment. Therefore, the syntax of the language of the interactive program was equally unfamiliar to all participants in the study. The syntax of the on-line editing language was a structured subset of English. Parsing was accomplished both positionally and by keyword recognition.

The first part of every command was a verb (RETRIEVE, DISPLAY, or CHANGE). The second parameter was a specifier indicating which and how many occurrences of the literal were to be considered, e.g., LAST 6, FIRST 1, EVERY, ALL. The literal came next and was identified by being enclosed between two system quotation marks (asterisks). The final command part was a file-range part indicating the portions of the file to be contextually searched or addressed, e.g., BETWEEN 10 AND 200 or IN 60. The two exceptions to this four-part syntax were (1) the DISPLAY command which required only the word LINES and a range of line numbers, and (2) the CHANGE command which of necessity included a search-target literal and a replacement literal.

An on-line tutorial program was embedded in the text-editor and had associated with it the EXPLAIN and HELP verbs.

Two different versions of the on-line text editor, an Inflexible version and a Flexible one, represented two levels of interface flexibility. In the Inflexible version, all command parts had to be spelled out fully. No mnemonics or abbreviations were permitted. Where a space was required as a delimiter, one and only one space was permitted. Every part of the syntax was essential for every command. Each statement had to be terminated with an end-of-statement character, predefined by the program to be a dollar sign.

Flexibility was operationally defined chiefly as the factors inherent in the interface program which would make it easier or more convenient for the user to express his commands, given that the commands had to follow a specific syntax. The design criteria specified that the user of the Flexible version should have as much freedom as possible in expressing his commands to the editor within the constraints of the language syntax, so long as any ambiguity as to user intent could be satisfactorily resolved by the program.

The syntax of the Flexible version was identical to that of the Inflexible with the following exceptions:

(1) the ability to reset the quotation mark and end-of-statement characters. A RESET command served this function (e.g., RESET QUOTES TO " or RESET EOS TO ;).
(2) the ability to abbreviate by truncation, so long as the remaining stem was unambiguous among other reserved words identified with that particular command part. The legal abbreviations for DISPLAY, for example, were: D DI DIS DISP DISPL DISPLA.
(3) the ability to omit certain optional command parts, the meaning of which was assumed according to predefined default values. The decision logic behind selection of default values was selection of those values which, in the designer's estimation, were the values which would be most frequently intended, so long as inadvertent omission of a command part would not result in assumption of an option which would be costly in terms of excessively long displays or catastrophic inadvertent modification of the file.
(4) the ability to declare equivalent synonyms for any reserved word in the language and subsequently to

abbreviate them according to the rules described in (2) above. The SYNONYM command was provided (e.g., SYNONYMS OF RETRIEVE ARE FETCH, FIND).

(5) the ability to use any number of spaces, commas, or periods as delimiters.

(6) the ability to use several equivalent forms of file-range parts. For example, these are all equivalent: BETWEEN 1 AND 10, BETWEEN 1-10, IN 1 . . . 10, and just the digits 1 10.

(7) the ability to omit an end-of-statement character without the command being aborted. Instead, an advisory warning message, to the effect that one would be inserted and processing would continue, was issued.

No differences between the two versions were apparent to the user until after completion of the first editing task according to the rules of the Inflexible version. Then, prior to presentation of each of the second through fifth tasks, a new interface flexibility option was offered to the user, along with instructions and examples of how to use it. The options were introduced in the following order: (1) resetting of quotation marks or end-of-statement characters, (2) abbreviations, (3) omission of optional command parts, (4) declaration of synonyms for any reserved word.

All instructions were administered on-line by a Computer-Assisted Instruction (CAI) module which taught very basic text-editing and the use of this particular text-editor. This module was identical for users of both versions. No prior knowledge of text-editing was assumed.

The experiment involved only one session of approximately two hours' duration. Upon, arrival at the CAI Laboratory at the University of Texas at Austin where testing was conducted a user was randomly assigned to one of four groups: Flexible/CRT, Flexible/Teletype, Inflexible/CRT, Inflexible/Teletype. All subsequent interactions were between the user and the computer. A proctor was available in case of system malfunction only. Events took place in this order: (1) introduction to the experiment and terminal familiarization, (2) demographic data collection, (3) typing practice and test, (4) survey of user attitudes toward the computer, using semantic differentials, (5) a five-item state anxiety scale from the State-Trait Anxiety Inventory,[3] (6) the 28-item hostility scale of the Multiple Affect Adjective Checklist,[4] the results of which will be reported in a later publication, (7) the CAI module.

The experimental task consisted of 18 subtasks which each required that an error in a computer-based text file be corrected using the on-line text-editor. The error was described to the user, who then had to formulate the text-editing command to accomplish the correction.

After completion of the tasks, the same anxiety, attitudinal, and hostility measures were repeated. Analysis of their results appears in Walther.[5]

RESULTS AND DISCUSSION

The principal data analysis employed a General Linear Models approach,[6] sometimes referred to as a multiple linear regression technique. Bennett[2] observed that controlled experimentation involving the user interface had produced results of questionable value because of the "unknown effect of uncontrolled variables on experimental subjects." This study capitalized on an important statistical capability of the General Linear Models approach—the ability to hold statistically constant, or fixed, all variables which were measured but which were not under experimental control, and whose effects were not specifically being investigated in a particular analysis. The net effect is the same as if experimental subjects had been selected such that they were all of the same sex, assigned to the same type of terminal device, had identical initial levels of anxiety and hostility, typed exactly at the same speed, shared in common the same amount of prior experience, and shared the same attitudes toward the computer.

In an experiment such as this, it would be obviously ideal if one could conclude that, irrespective of terminal type, sex, anxiety, attitude, typing ability, experience, or any other user attribute, it is better under all conditions to give the user a certain level of interface flexibility. However, as this study shows, things are not that simple. When the superiority of the Flexible version, for instance, depends on one or more factors other than flexibility itself, then it is said that the level of flexibility *interacts* with the one or more other variables, that they act in combination with each other to influence user performance.

Only those tests which were both statistically significant and relevant to this paper will be discussed. In all, there were fifty-five separate statistical tests made on the data.

Two measures of user performance are the criterion, or dependent, variables in this study: time-for-task and syntax-error-frequency.

TIME-FOR-TASK

User experience determines whether the flexible version is "better" with respect to time-for-task

Intuitively it was thought that users of the Flexible version would work faster during the editing tasks, irrespective of other factors, and that the more experienced users would be able to complete the tasks in proportionately less time.

Contrary to expectation, not all users of the Flexible version worked faster. Experience was a determinant of the level of interface flexibility which resulted in more rapid work. All users of the Flexible version worked faster than their similarly-experienced counterparts using the Inflexible version.

The more experience the user had with on-line systems, the better he was able to use the options to speed up his work. As expected, experience could not compensate for the non-availability of options in the Inflexible version, and inexperienced users worked at about the same rate with that version as those having considerable experience. Strangely enough, users of the Inflexible version and having *some* experience took longer than any other group to complete the editing tasks.

Flexibility options appear to help save time for all users except those having absolutely no prior experience with on-line systems. The Flexible version by design required less typing through the availability of abbreviations and the ability to omit optional parts. The more experienced users of the Flexible version were better equipped by virtue of their experience to take advantage of these short-cuts.

Users with no prior experience at all were probably somewhat overwhelmed with the options at a time when they had not yet become comfortable with the basic text-editing commands or with interactive computing in general. No explanation can be offered as to why the users of the Inflexible version and having *some* experience took so much longer than either of the other two Inflexible groups having none and much experience, respectively.

Terminal-type and "evaluative" attitude both determine whether the flexible version is "better" with respect to time-for-task

It was predicted that users of the same version and having the same prior experience level would work faster if they were using a CRT, even though it operated at the same speed as the teletypewriter. Although terminal type had an effect on time-for-task, the nature of its effect with respect to a particular level of flexibility was a function of the user's initial "evaluative" attitude (how much he liked the computer and how "good" he rated it). As attitude became less positive, times-for-task increased for the Flexible/CRT group. As initial attitudes became less favorable, times decreased in the Inflexible/CRT group. There was no relationship between time and attitudes among teletypewriter users of either version.

Why should the times-for-task of only the CRT users be related to their "evaluative" attitudes? Anecdotal information provided by the users upon conclusion of the experiment indicated that CRT users felt more positive about the computer than users of the teletypewriters. For Flexible/CRT users who initially felt extremely positive about the computer, the features of the Flexible version matched their expectations of computers and probably provided sufficient incentive for them to work against the possible obstacles of greater memory demands of the CRT terminal due to a lack of any hardcopy which can be referred back to, and the unfamiliar feel of the keyboard. Those whose attitudes were somewhat neutral may have been indicating that they just were not sure about computers and did not know what to expect. For them, the combination of this uncertainty with the presentation of the options at a time when they may have just begun to feel comfortable about using the system and the text-editor, and the greater memory requirements and keyboard physical features of the CRT, may have led them into more error conditions. These users also may have had to take longer between commands to figure out what to type next. Either of these conditions would have increased their times-for-task.

Those users of the Inflexible version at a CRT and who initially were most positively inclined toward computers probably experienced a mismatch between their expectations of the system and their observation of it, to the extent that they found working with the Inflexible version to be boring, highly repetitive, and consequently they had to take greater care to avoid errors. The silent characteristics of the CRT terminal did nothing to break the monotony for them. So it is possible that their need for more deliberate, careful typing is what increased their times. Those Inflexible/CRT users who expected less from the computer may not have been disappointed with the Inflexible version and may have been sufficiently fascinated with the CRT itself that the lack of flexibility in the interface was not particularly bothersome. It is somewhat surprising, however, that the attitudes had no apparent effect on time-for-task among teletypewriter users of either version.

SYNTAX ERRORS

Experience and "evaluative" attitude both determine whether the flexible version is "better" with respect to syntax-error-frequency

Prior to the experiment it was thought that increasing levels of prior on-line experience would result in progressively fewer syntax errors among users of the Flexible version. The reverse effect was anticipated among users of the Inflexible version. However, the data showed that experience, alone, was an insufficient determinant of the effects of interface flexibility on syntax errors. Initial attitudes also had a strong effect on the syntax error frequency of each of the three experience levels within the Flexible group. The user's predisposition or attitude seemed to have very little effect on the Inflexible group's syntax errors. Less favorable attitudes seemed to predispose all but very experienced users to greater error frequencies. Where attitudes were extremely favorable, error rates were quite low, with a slightly greater error rate being noted among users of the Flexible version. However, error frequencies rose sharply among these users as their attitudes were determined to be less favorable. The exceptions were high experience users of the Flexible version whose error rate, though very restricted in range by comparison with other users, tended to be higher where attitudes were the most favorable or positive. Those users who, on the basis of their brief encounter with the experimental on-line system, had developed highly favorable attitudes concerning the computer apparently liked the options subsequently given them in the Flexible version and were able to take advantage of these options without undue syntax errors. Those users having neutral or negative feelings about the computer and having none or only minimal prior experience seemed to be predisposed to making more syntax errors. Possibly, these users' lower expectations concerning the computer were consonant with their earlier discovery that the flexibility options could get them into trouble or would require greater mental effort until they achieved mastery in the use of the options.

It is possible that those in the Much Experience group, who also liked the computer very much, are the users who were encouraged to experiment and to "toy around" with the flexibility options. The journals of some of these people indicate this to be the case. They probably wanted to test the system, were possibly Computer Science majors at the university, and saw this as an opportunity for creativity. This kind of trial-and-error behavior would certainly result in a high incidence of syntax errors initially.

Irrespective of their attitude toward the computer, those totally inexperienced users in the Inflexible group made virtually no syntax errors at all. They had a version of the editor that was straightforward, easy to use, and simple to learn.

Their complete lack of experience and the fact that this was a "one-shot" encounter with the system gave them no basis for becoming either annoyed or frustrated, even in the face of the rigid constraints imposed on them by the Inflexible interface program.

Terminal-type and "evaluative" attitude both determine whether the flexible version is "better" with respect to syntax-error-frequency

Within each version of the editor, it was thought that users of the CRT terminals would make somewhat fewer errors than users of the mechanical teletypewriter terminals. However, within each version, CRT users made more syntax errors than the others, with a linear, positive relationship between syntax-error-frequency and "evaluative" attitude. As noted earlier, there was a very sharp increase in syntax-error-frequency among users of the Flexible version as these users' attitudes were observed to become less favorable. This was found to be true of users of both types of terminals. Therefore, attitude is a good predictor of syntax-error-frequency among users of the Flexible version, due to its high correlation with error-rate, but it is less helpful in predicting syntax errors of users without access to flexibility.

The chief differences between the two types of terminal devices are (1) the greater memory demand placed on CRT users with information at the top of the screen is erased as a new line is written at the bottom, (2) completely silent operation of the CRT compared to relatively noisy teletypewriters, (3) more modernistic-looking CRT consoles which may be perceived as being less mechanical in appearance, and (4) easy-to-press electronic action keyboards on the CRT and the mechanical linkage action of a teletypewriter. It is unclear which of these differences gave the CRT an inferior position with respect to user-performance. The fact that the teletypewriter gave the users access to all previous dialogue transcripts, the fact that there was a one-to-one correspondence between a key-press and an audible click as the character was printed (keeping the user from having to look up at the display to see if something happened), and the possibility that the action of the teletypewriter keyboard was more similar to the "feel" of manual typewriters with which undergraduates may have been more experienced, all probably helped the teletypewriter users to work faster and more accurately.

CONCLUSION

The results clearly indicate that interface flexibility is not uniformly effective with all users in optimizing performance. In a single encounter with the on-line system, users are more prone to make syntax errors if offered short-cut flexibility options. Nevertheless, most all users of the Flexible version worked significantly faster than those not having the options. The exceptions were the novices who worked more rapidly without the options than with them.

A user's prior on-line experience is a sufficient basis for deciding which interface to offer him if it is important to minimize length of an on-line session. If he has no prior on-line experience, he will work faster in his first session without options being offered. If he has any prior experience at all, he will need less time if offered flexibility options.

Neither a CRT nor a teletypewriter always workes better with any one level of flexibility in minimizing time-for-task. If only teletypewriters are available, there is no need to determine the user's "evaluative" attitude toward the computer in order to keep sessions as brief as possible. Using a teletypewriter, neither the presence nor absence of flexibility options will result in significantly different times. However, where a CRT is involved—even one that operates at typewriter speeds, the user's attitudes toward the computer become a critical factor in predicting time-for-task.

The experience factor alone is also an insufficient basis for determining whether flexibility will minimize syntax errors in a single session with the computer. In general, users having access to flexibility options made many times more syntax errors. However, the less experienced users who had favorable attitudes made very few errors.

If duration of a session is critical, flexibility can be very instrumental in facilitating more rapid work for almost all users. But, if the lack of syntax errors at a first session is the more important criterion, flexibility should be offered only to the less experienced users with very favorable attitudes, and to experienced users with a neutral or negative attitude toward the computer. The more experienced user with positive attitudes would be expected to make progressively fewer syntax errors in subsequent sessions as his temptation to experiment declined.

The data suggest that silent-keyboard terminal devices without hardcopy capability increase the likelihood of errors.

The one initial measure which never seemed to make any difference was typing ability, confirming Morrill's[7] findings that professional typing skills are not necessary for effective system use if the inputs are short and direct.

This study also confirmed the findings of Carlisle[8] that more errors are committed by users of a CRT than by teletypewriter users.

Future research in this area should involve multiple sessions by the same users to see how performance is affected *over time.*

Interface flexibility indeed appears to be a viable solution to the problem of giving the on-line interface the quality of adaptability. Options offered the user of an applications program in the manner in which he expresses his commands to the computer are not uniformly "good" for everyone's performance, and we now have a beginning basis for predicting those kinds of users for whom it is the best approach.

REFERENCES

1. Martin, J., *Design of Man-Computer Dialogues*, Prentice-Hall, Englewood Cliffs, 1973.
2. Bennett, J. L., "The user interface in interactive systems," in C. A. Cuadra (ed.), *Annual Review of Information Science and Technology*, Vol. 7, American Society for Information Science, Washington, 1972, pp. 159-196.
3. Spielberger, C. D., R. L. Gorsuch, and R. E. Lushene, *Manual for the State-Trait Anxiety Inventory*, Consulting Psychologist Press, Palo Alto, California, 1970.
4. Zuckerman, M., and B. Lubin, *Manual for the Multiple Affective Adjective Checklist*, San Diego, Educational and Industrial Testing Service, 1965.
5. Walther, G. H., *The User-Computer Interface: The Effects of Interface-Flexibility, Experience, and Terminal-Type on User Performance and Satisfaction*, unpublished Ph.D. thesis, The University of Texas at Austin, 1973.
6. Ward, J. H., Jr., and E. Jennings, *Introduction to Linear Models*, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
7. Morrill, C. S., C. Goodwin, and S. L. Smith, "User input mode and computer-aided instruction," *Human Factors*, 1968, *10*(3), pp. 225-232.
8. Carlisle, J. H., *Comparing Behavior at Various Computer Display Consoles in Time-Shared Legal Information*, Rand Corporation, Santa Monica, California, September, 1970 (AD 712695).

The control data STAR-100—Performance measurements

by CHARLES J. PURCELL

Control Data Corporation
St. Paul, Minnesota

INTRODUCTION

The CONTROL DATA STAR-100 (STring-ARray) Computer is a very large, general purpose, high speed computing system. The STAR-100 computer utilizes integrated circuitry, ferrite core memory, 400 hz power and freon cooling in the hardware implementation. The logical design of the computer combines stream processing, virtual addressing, hardware macro instructions, segmented (pipeline) arithmetic units and a 256-word high speed register file to perform arithmetic and logical operation on discrete or structured data elements (Figure 1).

Several of the STAR-100 Systems are in final test and checkout at this time preparatory to delivery. The STAR-100 is extending the state of computer art in terms of the memory bandwidth, the arithmetic pipeline performance, hardware aids to the operating system and built-in sub-routines for the programmer. As with any substantial improvement in computing, we at CONTROL DATA have had to learn a great deal about the use of this computer before it could be completed. The particular areas of substantial interest are the hardware design, potential performance and the program language environment.

HARDWARE DESIGN

The initial design goal of the STAR-100 required the production of at least 40 million additions per second on arrays of data formatted in 64-bit floating point words. This requirement in the STAR-100 is met by the execution of a three-address memory to memory array instruction. The source arrays and the result array must be carefully located in contiguous cells of memory in order to provide the logic designer with the power to completely overlap more than 80 million loads, 40 million adds and 40 million stores every second. The instruction repertoire of the STAR-100 was carefully selected to match the logic design with the program. One can visualize the demands on the programmer by considering the STAR-100 as a simplified hardware implementation of a programming language.[1]

Array operations are provided in the STAR-100 on either 32-bit or 64-bit floating point operands. Consideration of the total computational requirements suggested that the rate of array operations would require support by a wide variety of textual operations on byte strings as well as logical operations on bit strings. The 256 word register file is used to contain scalars, descriptors or index quantities used in support of the array or string operations. Branch and test operations are available on bits, bytes indices, scalars, arrays, strings or conditions. In all, some 230 instructions are provided in the STAR-100.

A physical design evolved from these considerations (Figure 2) whereby the complete computer is assembled in one large monolithic structure. The minimum STAR-100 computer contains 4 million (8-bit) bytes of memory, 4 I/O channels, 1 direct access channel, the Central Processor and a Maintenance Control Unit. The Central Processor contains the Storage Access Control unit (SAC), the Stream Control unit, and the floating point pipelines.

The Stream Control unit of the Central Processor contains all streaming and instruction control, and operand alignment, buffering and addressing. The Stream Control unit also contains the register file of 256 64-bit words used for instruction and operand addressing, indexing, storing constants, field length counts, and as source and destination for conventional register to register instructions in 3-address format. A microcode memory in the Stream Control unit controls the initiation, interrupt (if necessary) and termination of all array and string instructions.

The register file utilizes a twenty nanosecond read orwrite cycle time semi-conductor memory component. The microcode memory utilizes an eighty nanosecond, two phase, semiconductor memory. Main memory of the STAR-100 is built of thirty-two banks each of 1.28 microsecond full cycle time, 512 bit word, ferrite core memory. At this time, the checkout process is focusing on the verification of each individual instruction to take over control of the data streams, correctly operate and form the results in the appropriate location. Then each and every instruction combination must also be verified as to the correct control and results.

The Central Processor of the STAR-100 System is aided by a number of auxiliary processing input/output stations (Figure 3). These stations provide intelligent control of the input/output requirements of the STAR-100 by use of control messages from the central monitor program. These con-

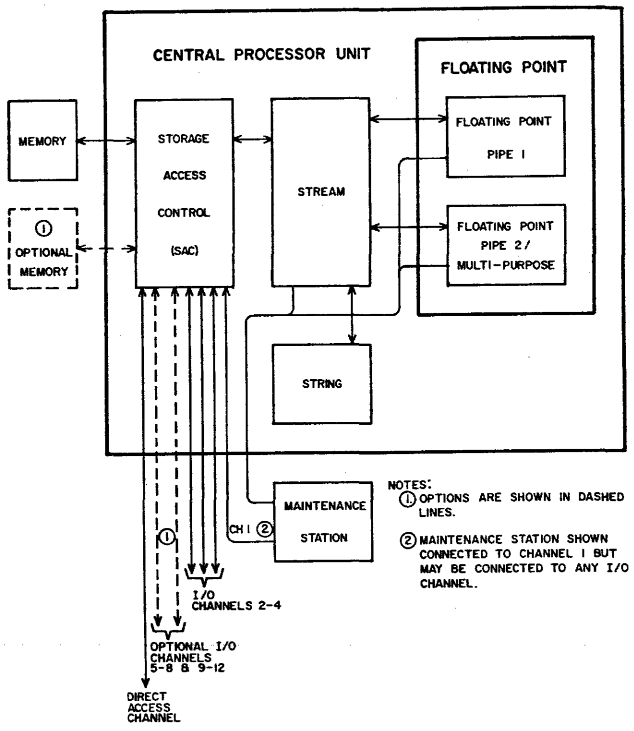

CENTRAL PROCESSOR UNIT
FLOATING POINT
MEMORY
STORAGE ACCESS CONTROL (SAC)
STREAM
FLOATING POINT PIPE 1
FLOATING POINT PIPE 2/ MULTI-PURPOSE
OPTIONAL MEMORY
STRING
MAINTENANCE STATION
CH 1
I/O CHANNELS 2-4
OPTIONAL I/O CHANNELS 5-8 & 9-12
DIRECT ACCESS CHANNEL
NOTES:
① OPTIONS ARE SHOWN IN DASHED LINES.
② MAINTENANCE STATION SHOWN CONNECTED TO CHANNEL 1 BUT MAY BE CONNECTED TO ANY I/O CHANNEL.

Figure 1—Basic CDC STAR-100 configuration

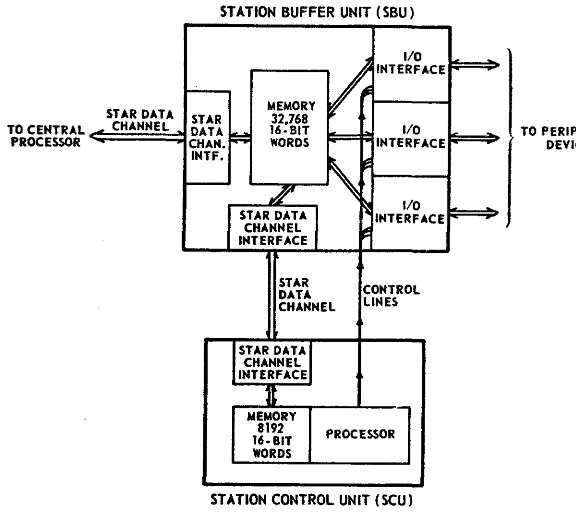


Figure 3—Basic station

trol messages are processed by the station control unit (processor) while data streams move into and out of the station buffer unit.[2] A typical STAR-100 system is shown in Figure 4. Peripheral systems of this type have been in use for several years in our laboratory.

PERFORMANCE MEASUREMENTS

As of December 1973 about ⅔ of the instruction vocabulary of the STAR-100 has been verified for correct performance on either of two central processors. This verification includes the extensive range of interrupt possibilities as well as most of the instruction combinations. As a result of these tests we have verified the raw performance potential of the STAR-100 via the contract SAMPLE PROBLEM. This test is representative of the best use of the STAR-100 in forming arithmetic results on packed arrays of data in either 32-bit or 64-bit floating pointing format. Some 32 billion intermediate results are correctly formed in 625 seconds yielding a rate of greater

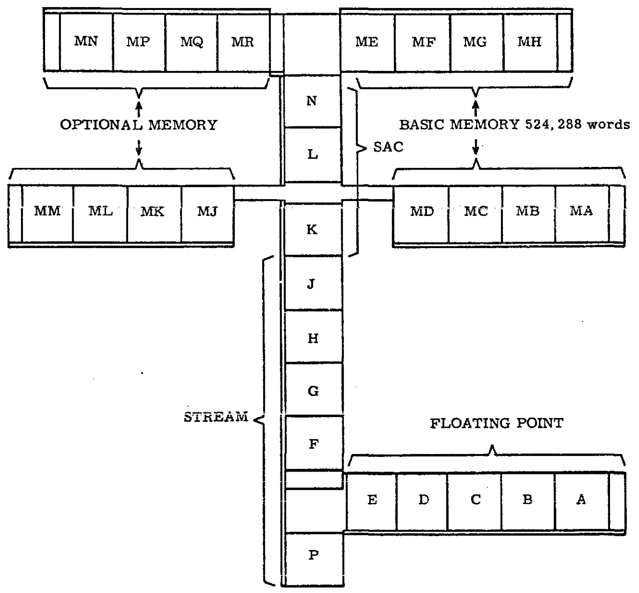


Figure 2—STAR-100 section identification

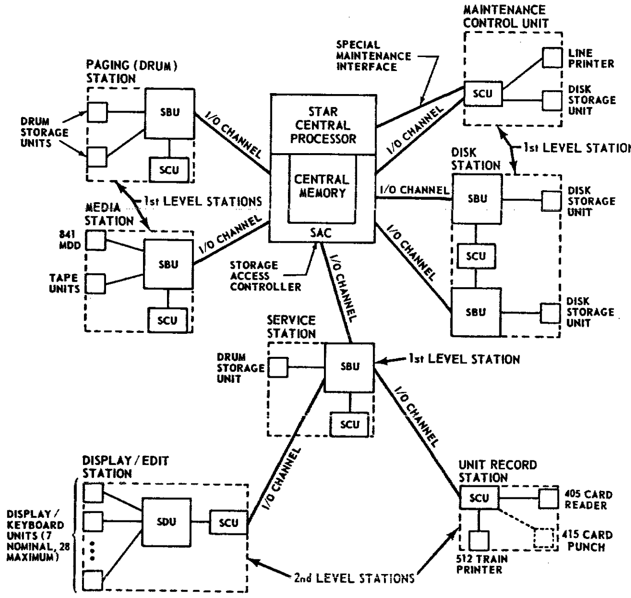


Figure 4—Typical STAR-100 system

than 50 million results per second. Other meaningful performance measurements await the installation of the STAR Operating System and appropriate compilers, interpreters and assemblers on the STAR-100 (MARCH-MAY 1974).

The performance potential of the peripheral system has been reported in the previous NCC.[2]

SOFTWARE REQUIREMENTS

The general purpose instruction repertoire and control facilities of the STAR-100 were designed to facilitate the development of dynamic recursive languages such as PL/1, ALGOL and APL. Users, in general, have continued to use FORTRAN practices in code development as well as for execution time service routines. Thus the STAR-100 System will require the implementation of FORTRAN (ANSI 1973) with substantial enhancements before the official release date in 1974. The planned enhancements to FORTRAN can be characterized as explicit APL-like monadic and dyadic functions in a static pre-defined environment. The expansions include:

1. Subscripting (K, I, L, M, N, J, S)
   implied DO loop (K, I, L, M, J, BEGIN:END: DELTA)
   cross section (K, I, *, M, N, J, S)
   offset (K, I, L, M, N, J, S; LENGTH)
   where any construct that specifies operations on an ARRAY will form a VECTOR instruction; other constructs will be executed in the conventional instruction sequence.
2. New Data Types
   BIT one bit per bit of memory
   SPARSE an order vector BIT mapping of an ARRAY
   DESCRIPTOR a scalar containing the length and base address of an ARRAY or CHARACTER
3. ARRAY INTRINSICS and BASICS
4. ARRAY FUNCTIONS
   where an ARRAY argument is processed to return an ARRAY result.
5. Selection Operations

controlled store	C=B. CNTRL. A
compress	C=B. COMPR. A
expand	C=B. EXPAND. A
gather	C=I. COMPR. A
scatter	C=I. EXPAND. A

   where A is an ARRAY, B is BIT, C is the result, and I is an Indexlist.
6. ARRAY ARITHMETIC
7. ARRAY TESTS

compare	B=A. RELATIONAL. C
search	I=A. RELATIONAL. C
select	

The general theme of these expansions is to produce a language super-set which could be compiled and executed on a conventional computer having the suggested language constructs in its compiler. FORTRAN remains a language fixed at compile time with an even larger set of reserved words.

Code optimization for the STAR-100 computer is a rich and rewarding field in terms of potential performance improvements. The compiler must allocate the use of the register file, eliminate common sub-expressions, minimize redundant address calculations and exploit the special hardware facilities of the STAR-100. In many cases a conventional FORTRAN program must be de-compiled in order to determine the intent of the programmer (INNER PRODUCT, INITIALIZATION, etc.).

The same expansions to FORTRAN would be required for the STAR implementation of PL/1 or ALGOL. APL makes full use of the STAR-100 via APL*STAR.

CONCLUSION

This project required the services of many divisions in CONTROL DATA, particularly the Advanced Development Laboratory. Many of the concepts utilized in the STAR-100 have been supplied by the programming community in general. These concepts were built into the STAR-100 in response to the question: "Just what is it that these programmers do?"

REFERENCES

1. Iverson, K. E., *A Programming Language*, Chapter 1, pp. 1-40, Wiley, 1962.
2. Hohn, W. C., and P. D. Jones, "The CONTROL DATA STAR—100 Paging Station," Vol. 42 N.C.C. Proceedings 1973, pp. 421-426.
3. Control Data, *STAR—100 Features Manual*, Pub. No. 60418100 October 1973, Control Data Corp., St. Paul, Minn. 55112.

Operational experiences with the TI Advanced Scientific Computer

by W. J. WATSON and H. M. CARR

Texas Instruments Incorporated
Austin, Texas

INTRODUCTION

Since 1966 a large computer development program has been conducted by Texas Instruments. The goal for this effort was to provide needed capacity for supporting seismic processing, plus offering a general purpose capability for large scientific problems.

This development has resulted in the Advanced Scientific Computer (ASC)—a highly modular system offering a wide spectrum of processor power, memory sizes, and I/O capability. The ASC is a high-speed, large-scale processing system featuring extensive use of pipelining, multiple arithmetic units, separate control processors, large and fast central memory, and extensive user software aids. The central processor has both scalar and vector instruction capabilities.

First delivered in 1972 and placed into operational status during 1973, several operational ASC systems now offer extremely high processing rates for particular classes of problems.

OVERVIEW OF THE SYSTEM

The major subsystems of a typical configuration are shown in Figure 1: the central memory, the central processor, the peripheral processor, on-line bulk storage, a digital communications interface, plus a selection of standard peripherals.

The peripheral processor has been designed for executing the operating system. The central processor has been designed expressly to provide high computing speeds when operating upon large arrays of data. The central processor operates as a slave to the peripheral processor. This design approach was chosen to maximize the overlapping of system overhead tasks with the execution of user programs. In operation the job stream is analyzed by the peripheral processor. The language processors, plus user object code, are executed by the central processor. System control and I/O tasks are processed by the peripheral processor. I/O is routed through high-speed, head-per-track disc storage. A data communications interface for the common carriers is provided for the support of remote batch and interactive terminals. Standard types of peripherals are also provided. The central memory serves as the common communications and access storage medium for these subsystems.

CENTRAL MEMORY

The ASC central memory consists of a memory control unit (MCU) and appropriately sized modules of high-speed or medium-speed central memory. Optionally, a medium-speed central memory extension can be used in conjunction with a high-speed memory.

The MCU is organized as a two-way, 256-bit/channel (8-word) parallel access traffic net between eight independent processor ports and nine memory buses, with each processor port having full accessibility to all memories. The nine memory buses are organized to provide eight-way interleaving for the first eight buses with the ninth bus used for the central memory extension. The MCU provides the facilities for controlling access from the eight processor ports to a CM having a 24-bit address space (16 million words). A port expander can be utilized to expand the number of processor ports. Figure 2 illustrates this structure.

The semiconductor high-speed central memory modules have a cycle time of 160 ns and a read time of 140 ns. Additionally, all transfers are 256 bits (eight 32-bit words) with a Hamming code providing single-bit error correction and double-bit error detection for each 32-bit word. High-speed central memory is typically divided into eight equal-sized modules which allow for eight-way interleaving.

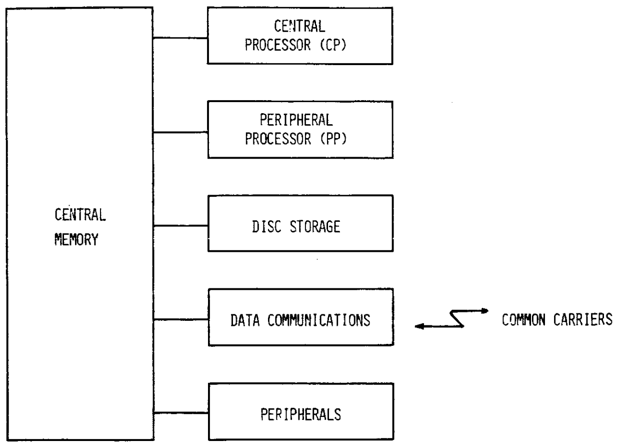


Figure 1—Major ASC subsystems

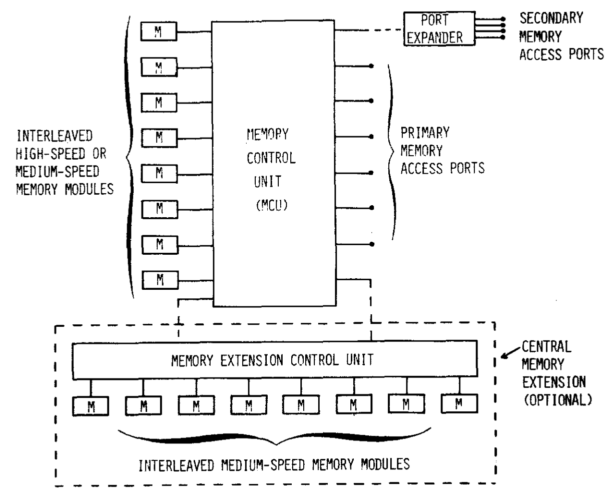

PORT EXPANDER
SECONDARY MEMORY ACCESS PORTS
INTERLEAVED HIGH-SPEED OR MEDIUM-SPEED MEMORY MODULES
M
MEMORY CONTROL UNIT (MCU)
PRIMARY MEMORY ACCESS PORTS
MEMORY EXTENSION CONTROL UNIT
CENTRAL MEMORY EXTENSION (OPTIONAL)
INTERLEAVED MEDIUM-SPEED MEMORY MODULES

Figure 2—Modular structure of the ASC central memory

The optional central memory extension allows large amounts of medium speed memory (1 μs semiconductor technology) to be used in the normal address space of central memory. Block transfer between memory extension and high-speed memory is controlled by the peripheral processor and will transfer at a rate of 40 M words per second.

Memory mapping registers and protection registers are used to facilitate central memory management and access control of the ports.

CENTRAL PROCESSOR

The central processor provides both scalar (single operand) and vector (array) instructions at the machine level. The basic instruction size is 32 bits, with 16-, 32-, or 64-bit operands. The single instruction stream, which contains a mixture of scalar and vector instructions, is preprocessed by the instruction processing unit.

The central processor design is such that one, two, three, or four execution units or "pipes" can be provided. These units employ the pipeline concept in both scalar and vector modes. A single execution unit can have up to twelve scalar instruction in process at one time. From one to four vector results can be produced every 60 ns, depending on the number of execution units provided.

The CP has 48 program-addressable registers. This group of 32-bit registers consists of sixteen base address registers, sixteen arithmetic registers, eight index registers, and eight vector parameter registers. This last group is used to extend the instruction format for the complete specification of vector instructions.

The CP scalar instruction repertoire includes an extensive set of load and store instructions: halfword, fullword, and doubleword instructions, with immediate, magnitude, and negative operand capabilities. Ability to load and store register files and to load effective addresses is also available. Arithmetic scalars include various adds, subtract, multiply, and divide for halfword (16-bit) and fullword (32-bit) fixed point numbers and fullword and doubleword (64-bit) floating point numbers. Scalar logical instructions are provided as are arithmetic, logical, and circular shifts. Various comparison instructions and combination comparison-logical instructions are provided for halfword, fullword, and doublewords. Many combinations of test and branching instructions with incrementing or decrementing capability are also available. Stacking and modifying arithmetic registers can be done with single instructions. Subroutine linkage is accomplished through branch and load instructions. Format conversion for single and doublewords, as well as normalize instructions, are available.

The vector capabilities of the CP are made available through the use of VECTL (vector after loading vector parameter file) and VECT (assumes parameter file is already loaded) instructions. The vector repertoire includes such arithmetic operations as add, subtract, multiply, divide, vector dot product, matrix multiplication, and others for both fixed point and floating point representations. Vector instructions are also available for shifting; logical operations; comparisons; format conversions; normalization; and special operations—such as Merge, Order, Search, Peak Pick, Select and Replace, among others.

One important characteristic of the vector instruction capability is the ability to encompass three dimensions of addressability within a single vector instruction. This is equivalent to a nest of three indexing loops in a conventional machine.

The basic structure of the CP, shown in Figure 3, has three major components: the instruction processing unit (IPU) for non-arithmetic stages of instruction processing for the CP instruction stream, the memory buffer unit (MBU) to provide operand interfacing with the central memory, and an arithmetic unit (AU) to perform the specified arithmetic or logical operations. Figure 3 shows a CP diagram for 2- or 4-pipeline CP's, each with a corresponding number of MBU-AU pairs. Note that a memory port is required for the IPU and, in addition, one memory port for each pipeline (MBU-AU pair) in a CP.

A significant feature of the CP hardware is an operand look-ahead capability which causes memory references to be requested prior to the time of actual need. Double buffering

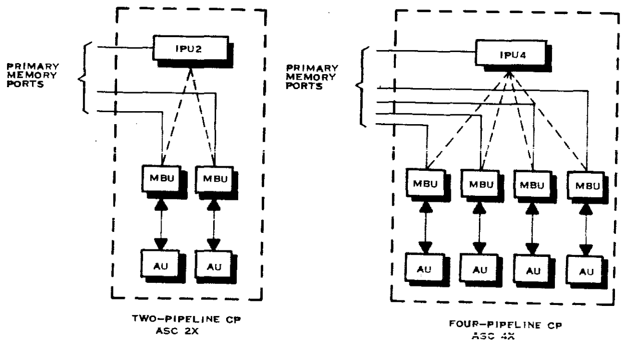


Figure 3—Basic structure of the CP

in multiple 8-word (octet) buffers for each pipeline provides a smooth data flow to and from each arithmetic unit. The pipelined AU achieves its highest sustained flow rate in the vector mode, typically a result each 60 ns per AU, or an average of 15 ns per result for a 4-pipe central processor.

Instruction processing unit

The primary function of the instruction processing unit (IPU) is to supply a continuous stream of instructions for execution by the other parts of the CP. One Central Memory port is required to provide the instruction stream. Two 8-word (octet) buffers are utilized to achieve a balanced stream of instructions from memory to the IPU. Instructions are transferred from memory in octets as are all other references to memory for fetching or storing of information.

Up to 36 instructions in various stages of execution can be overlapped within the 4-pipe CP. There are twenty positions for instructions in the 2-pipe CP and twelve positions for instructions in the 1-pipe CP. Four levels are contained within the IPU, and eight levels are contained in each arithmetic pipeline (MBU-AU pair). The IPU performs routing of instructions to the MBU-AU pairs based on an optimum use of arithmetic unit capability.

Vector processing is altered by software in order to distribute segments of the vector for multiple pipe systems.

Several features are provided to alleviate the potential problems of branches and instruction dependencies in the instruction pipeline.

Memory buffer unit

The memory buffer unit (MBU) provides an interface between central memory and the arithmetic unit. Its primary function is to supply the arithmetic unit with a continuous stream of operands from memory and to provide for the storing of the results back to memory. All references to memory, whether for fetching or storing, are made in 8-word increments (octets).

The MBU has three double buffers, one octet per buffer, called the "X" and "Y" buffers for input and the "Z" buffers for output. This double buffering is provided so that pipeline processing can be sustained at a high rate with minimal memory access conflicts.

Arithmetic unit

The primary function of a CP arithmetic unit (AU) is to perform the arithmetic operations specified by the operation code of the instruction currently at the AU level. There is one AU per pipeline in the CP, each having a 60 ns basic cycle time. A distinguishing feature of an AU is the pipeline structure which allows efficient execution of the arithmetic part of all instructions. There are eight exclusive partitions of the AU pipeline involved, each of which can provide an output every 60 ns. These eight sections are (1) receiver register,

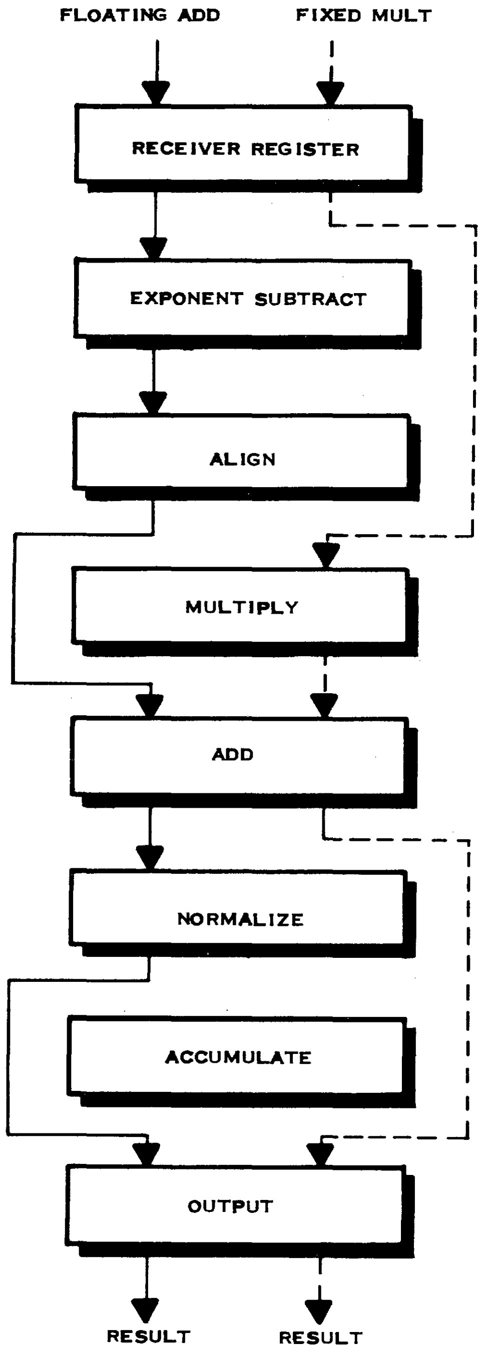


Figure 4—Arithmetic unit pipeline

(2) exponent subtract, (3) align, (4) add, (5) normalize, (6) multiply, (7) accumulate, and (8) output. Figure 4 shows how different sections of the AU are utilized for execution of particular instructions; i.e., floating point addition and fixed point multiplication.

An AU is a 64-bit parallel operating unit for most scalar and vector instructions. Exceptions are double length multiply and all types of division. In these circumstances various combinations of the components of the AU are utilized; and, therefore, more than one clock cycle is required to complete these arithmetic operations.

THE PERIPHERAL PROCESSOR

The peripheral processor (PP) is a powerful multiprocessor designed to perform the control and data management functions of the ASC. Several aspects of the implementation of the peripheral processor concept greatly increase the effectiveness of the ASC system.

The PP is a collection of eight individual processors called virtual processors (VP's). Each VP has its own program counter along with arithmetic, index, base, and instruction registers. The eight VP's share a read only memory, an arithmetic unit, an instruction processing unit, and a central memory buffer. Use of the common units is distributed among the VP's using sixteen single 85 ns cycles. When an equally distributed sequence of time units is used, each of the eight VP's receives two 85 ns cycles every 1.4 μs. The typical PP instruction requires two 85 ns cycles for completion. The distribution of available time units can be dynamically varied to suit particular processing requirements.

The 4K 32-bit words of read only memory within the PP is utilized for program storage and execution of those short routines which are highly utilized by the VP's, such as polling loops.

Because the PP is intended to perform control functions rather than execute mathematical algorithms, the instruction set is oriented toward control operations and does not require multiplication, division, or floating point operations. The instruction format is similar to that of the central processor, using a 32-bit word for each instruction. Instructions are provided for bit (1 bit), byte (8 bits), halfword (16 bits), and fullword (32 bits) operations.

Each VP has direct access to the entire central memory for program execution and data storage. Therefore, a single copy of reentrant code can be executed simultaneously by more than one VP.

The communications register (CR) file contains sixty-four 32-bit word registers which are program addressable by the VP's. The CR file serves as the principal storage media for control information necessary for the coordination of all parts of the ASC system.

DISC STORAGE

Disc storage is the principal secondary storage system for the ASC system. Disc storage consists of head-per-track (H/T) disc systems supplemented by positioning-arm disc (PAD) systems.

The H/T disc system is a high-performance device whose effective performance is further enhanced because the operating system utilizes a shortest-access-time-first (SATF) algorithm for data transfers. This combination of hardware and software provides a very high effective transfer rate. Each H/T disc module has a capacity of 25 million 32-bit words with a transfer rate of approximately 500K words per second. Using the shortest-access-time-first algorithm, access time will average approximately 5 ns which results in an exceptionally fast "effective" transfer rate.

DATA COMMUNICATIONS

The data communication system is very modular and, thus, externally flexible in the various devices which may be utilized for communication with the ASC. Data communications are controlled by a data concentrator which, in turn, interfaces to the MCU through a channel control device.

The data concentrator is a TI-980A minicomputer equipped with special-purpose hardware communication interface units on its direct memory access ports.

The data communications system presently supports communication with three types of stations: high-performance user terminals, other large computers, and remote concentrators. The system can be easily extended to support smaller terminals down to the teletype level. These stations may be either remote or local.

Remote links are presently implemented with non-switched, full duplex common carrier data transmission facilities. Data is transferred over these links synchronously at rates determined by the modems and common carrier bandwidths. The data communication system supports transfer rates up to a maximum of 240,000 bits per second.

PERIPHERALS

Standard types of magnetic tape drives, card equipment, and printers have been interfaced with the ASC. These interfaces attach to primary or secondary memory ports through a variety of standard selected and multiplexed data channels. A subset of the system's peripherals can also be interfaced via the communications register file.

SYSTEM SOFTWARE

Software design and development for the ASC system has progressed in parallel with development of the hardware. This was accomplished through the use of simulators, meta-assemblers, and higher level programming languages implemented on the systems supporting Texas Instruments' Corporate Information Center. Thus, the first version of this software was placed into operational status with the ASC prototype machine. The major software capabilities are discussed in the next few paragraphs with emphasis being given to those attributes which provide comprehensive and flexible programming facilities for the user.

ASC Fortran language

The most obvious interface between the ASC system and a user is with the translation of the user-written program into machine level instructions that efficiently utilize the special hardware features in the system. Texas Instruments has attempted to make this interface a smooth one by effort invested in compiler techniques. The result of this effort is the ASC NX Compiler, a highly optimizing, user oriented, software package that will produce code acceptable to a central processor with one, two, three or four pipelines (arithmetic units).

The ASC's Fortran language is an extension of ANS Fortran. The added language features permit the ASC Fortran programmer to define and use subarrays, cross-sections of arrays or subarrays, array assignment statements, and array intrinsic functions. This is not to provide unique access to hardware features, but to simplify the programming required for complex problems.

The ASC Fortran compiler was designed to meet the demands of the professional programmer. Its primary function is to translate Fortran code into object code which will execute the program in the shortest possible time. Because the ASC has both scalar and vector instructions, the compiler has the capability to recognize array-oriented operations specified in standard Fortran and to generate the equivalent vector instructions to perform the required operations. To provide the programmer direct access to the specialized vector instructions, array intrinsic and array generation intrinsic functions are provided.

The ASC Fortran compiler produces highly optimized object code with complete diagnostic analysis and messages. In general, the optimizing task is accomplished by performing optimization on the source program logic and on the object code instructions produced. Vector instructions are used where feasible. Scalar operations are reordered wherever possible without affecting results, so as to minimize both pipeline and memory reference delays. In addition, the compiler provides a complete set of informative messages regarding applied optimization procedures and where source program logic prevents optimization.

The optimizing algorithms encompass such areas as conventional optimization, instruction scheduling and vector generation with optimization.

Mathematical library

The ASC Mathematical Subprogram Library is unique in that it uses both scalar and vector capabilities. The scalar function subprograms include all of the single and double precision functions traditionally provided in Fortran libraries. In particular, it contains all of the ANS Fortran mathematical functions and all of the IBM S/360 Fortran mathematical functions. The vectorized math function subprograms exploit the vector instruction set of the ASC. A single call to a vectorized math function subprogram causes that function to be evaluated for the entire vector of arguments. The evaluation is effected by a sequence of vector instruction executions.

Both the scalar and the vectorized math function subprograms can be used by the Fortran and assembly language programmer. The Fortran compiler employs the vectorized math subprograms to replace multiple calls to a scalar subprogram when possible; however, this action may be overridden by the use of a Fortran compiler specification option.

Assembler

The ASC Assembler is a meta-assembler or translator which facilitates symbolic coding of the ASC Central or Peripheral Processors at the instruction level.

Linkage editor

The ASC Linkage Editor creates a load module for execution by linking separately assembled or compiled object modules obtained from the job input stream, user libraries or system libraries. Linking is accomplished by relocation, by resolving external references, and by allocating virtual memory.

Job specification language

The Job Specification Language (JSL) is a user-oriented language. It allows the user to specify the programs to be executed, the data files to be made available, the dependencies, if any, between individual programs of a job, and various cataloging and data management functions which may be specified. The user may specify and control a job without detailed knowledge of the Operating System. Wherever possible, default conditions have been built into this language so that only a minimum specification need be given by the user.

The Job Specification Language is composed of job definition statements, program processing statements, file processing statements, cataloging statements, and macro definition statements. It is an extendible (macro facility), programmable specification language rather than a set of control cards. The philosophy has been to provide many explicit statements with relatively few parameters for each, rather than a few statements with many operand fields that provide all functions.

The language provides JSL variables which allow the programmer to pass control information to and among CP programs at execution time. JSL control statements can be used to test these variables to determine the programs to be executed next. An executing job can initiate a deferred job; the decision to do so could be based on the value of a JSL variable within the executing job.

Operating system

The ASC General Purpose Operating System (GPOS) schedules and allocates system resources in response to user service requests in a multiprogramming environment. GPOS provides input/output service, data transfer within the system, file management services, and other system services in a straightforward manner. The utility and accessibility of the Central Processor to user programs is increased by

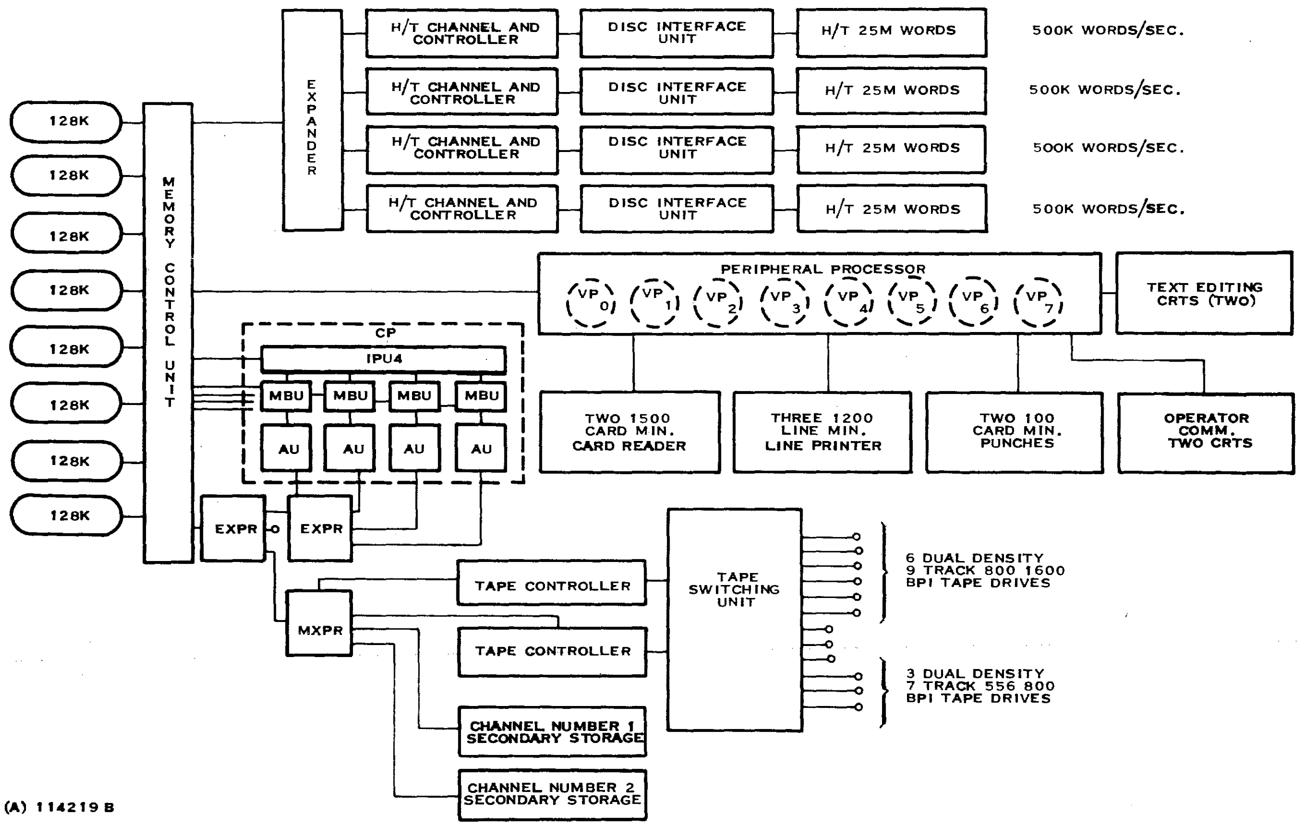

H/T CHANNEL AND CONTROLLER
DISC INTERFACE UNIT
H/T 25M WORDS
500K WORDS/SEC.
EXPANDER
128K
MEMORY CONTROL UNIT
PERIPHERAL PROCESSOR
VP 0
VP 1
VP 2
VP 3
VP 4
VP 5
VP 6
VP 7
TEXT EDITING CRTS (TWO)
CP
IPU4
MBU
AU
TWO 1500 CARD MIN. CARD READER
THREE 1200 LINE MIN. LINE PRINTER
TWO 100 CARD MIN. PUNCHES
OPERATOR COMM. TWO CRTS
EXPR
MXPR
TAPE CONTROLLER
TAPE SWITCHING UNIT
6 DUAL DENSITY 9 TRACK 800 1600 BPI TAPE DRIVES
3 DUAL DENSITY 7 TRACK 556 800 BPI TAPE DRIVES
CHANNEL NUMBER 1 SECONDARY STORAGE
CHANNEL NUMBER 2 SECONDARY STORAGE
(A) 114219 B

Figure 5—GFDL ASC configuration

GPOS performing all overhead functions in the Peripheral Processor. The operating system isolates the control, scheduling, and resource allocation algorithms for ease in "tuning" the system to match the specific requirements of each installation. The overall system architecture is maintained to accommodate hardware and software system growth and flexibility. GPOS, by its simplicity and modular design, minimizes the system use of central memory with a small resident system and the remainder of the system non-resident.

The design of GPOS exploits hardware features unique to the ASC. Most important of these features is complete access to central memory by the PP. Thus, a single reentrant copy of code is available to all processors; and, only a branch instruction is needed to switch a Virtual Processor from one function to another. The Communications Register (CR) file is used to allow one VP to control the other seven, while common access to the rest of this file supports communication between the processors and other system components.

OPERATIONAL HISTORY

The prototype ASC initially completed its checkout during the Spring of 1971. The system (Serial #1) was available for use as a software development tool and for customer demonstrations for the remainder of 1971. In 1972 the prototype was moved to a permanent location at the TI facility in Austin. During the period of downtime, a retrofit of the hardware was carried out to incorporate the latest version of circuits and boards and to support a production environment. System 1 was operational early in 1973 and is currently being devoted to software development and support of application program conversion to the ASC.

ASC #1 is configured with a one-pipe central processor, 128K words of high-speed central memory, 128K words of memory extension, a complement of head-per-track disc storage, a data communications interface, plus standard tape and paper devices.

Experience with an ASC operating in a center devoted to seismic production work is currently being gained in the TI facility at Amstelveen, Holland. This system (Serial #2) was delivered early in 1973 and essentially duplicates the capabilities described for the prototype machine. Additionally, several seismic interactive terminals are interfaced both locally and remotely to this system.

Seismic operational requirements are characterized by large data bases, much magnetic tape input and output, many job steps composed of long computational sequences, and the need to precisely control a complicated series of such jobs. In addition to the high computational speeds available on the ASC, the seismic center experience is showing that other ASC features are valuable when applied to this application.

Head-per-track disc storage, management of the data bases and scheduling by the dedicated virtual processors, and job control available via the JSL language appear to match the environment of seismic work. Applications programs are written in standard Fortran, and no need has been found to supplement the available compiler optimization by additional hand coding. The system is well supporting the requirements by generating significant improvements in unit processing costs, and by permitting new processing technologies to be economically feasible. Improved productivity of geophysicists and geologists through real-time interactive sessions is being achieved. It is expected that the use of ASC for seismic processing capacity will continue to grow at a rapid rate.

Operational experience has also been gained from the application of the ASC to the U.S. Government data-processing problem of ballistic missile defense. Serial #3, a one-pipe ASC with a configuration similar to the previously described systems, was delivered to the U.S. Army in the Summer of 1973. It is to be used for research into processing techniques employed in ballistic missile defense.

Application to long-range prediction of the earth's weather is the intended use of the largest and fastest ASC to be built to date. The National Oceanic and Atmospheric Administration (NOAA) has contracted for an ASC (Serial #4) for its Geophysical Fluid Dynamics Laboratory at Princeton University. Delivery is scheduled for early in 1974. The ASC is configured with a four-pipe central processor, one million words of high-speed central memory, head-per-track disc, text editing terminals, two channels of high density secondary storage devices, and standard magnetic tape and paper devices. This configuration is illustrated in Figure 5. Much experience has been gained using benchmark programs derived from weather models and the actual weather prediction codes themselves. Emphasis has been upon Fortran code generated by analysts and weather scientists instead of hand-optimized machine language. Results obtained from the system while undergoing final checkout at TI's facility showed the speeds available to be several times faster than other current computer systems.

For weather codes characterized by large data bases that are updated frequently, sequences of heavy computational work using the data, and mathematical operations performed on long arrays of data, the ASC is proving to be a valuable asset. The large central memory enables one to maintain ample data so that the central processor is utilized to a very high degree. The I/O and multiprogramming capabilities managed by the operating system resident in the peripheral processor also support high CP workloads.

TABLE I—Simple Examples of Vectors

```
(1)     DO    10    K=1, 50
        DO    10    J=1, 50
        DO    10    I=1, 50

   10       Z(I, J, K)=X(I, J, K) * Y(I, J, K)

(2)     Z=X * Y

(3)     VECTL (#460, B2)   VMF
```

TABLE II—Vector Instructions Produced from Weather Code

```
(1)            DO     100  K=1, 10
               DO     100  I=1, 144

               TBXY(I, K)=(T(I+1, K, J)+T(I, K, J)) * 0.5
               TXY(K, K)=(T(I+1, K, J)-T(I, K, J)) * RDX(JC)
               PBXY(I, K)=(PS(I+1, K, J)+PS(I, K, J)) * 0.5
         100   PXY(I, K)=(PS(I+1, K, J)-PS(I, K, J)) * RDX(JC)

(2)              VECTL (#3B8, B2)          VAF
                 VECTL (#3C0, B2)          VMF
                 VECTL (#3C8, B2)          VSF
                 VECTL (#3D0, B2)          VMF
                 VECTL (#3D8, B2)          VAF
                 VECTL (#3E0, B2)          VMF
                 VECTL (#3E8, B2)          VSF
                 VECTL (#3F0, B2)          VMF
```

MAXIMIZING PERFORMANCE

Experience thus far has shown that for the applications that have been considered by ASC users the most cost-effective performance is realizable when the capabilities of ASC Fortran and the optimizing compiler are used. Although particular sequences of code can be found wherein hand coding will improve the speed of execution, for the broad range of programs where much applications code is involved, compiler-generated object code is the best choice. American National Standard Institute (ANS) Fortran is completely sufficient, and vector instructions are readily produced from this Fortran. ASC extensions to the Fortran are sometimes found to be useful, not to provide unique access to some hardware feature but to simplify notation involved in writing the program so that the programmer can deal more directly with the mathematics of the application.

The ASC system design allows easy user access to performance enhancement through the use of additional central processor "pipes." Compiler software is responsible for both the generation of vector instructions and the partitioning of these vector operations over multiple pipes. Protection of the user from vector hazard conditions is carried out by the compiler. Partitioning of scalar instructions for multiple pipes is carried out by the CP hardware. Extensive checks are made by hardware to protect the user from illegal scalar conditions that might occur. For mixtures of vector instructions and for mixtures of scalars and vectors, the compiler prevents illegal conditions by the use of directive instructions for the CP to operate in either parallel mode (FORK) or sequential mode (JOIN). Thus, the burden is on the system instead of the user. Programs compiled for one-pipe ASC's will execute correctly on multiple-pipe systems. Performance will be increased via a recompilation for the multiple-pipe machine.

Some typical examples of efficient code produced from present applications will illustrate the optimization level provided by the system. Table I shows the type of instruction generated by the compiler from a typical triple-nested DO LOOP.

(1) gives the Fortran source with three levels of indexing,
(2) is an alternate notation that could be used, and
(3) is the single vector instruction produced.

TABLE III—ASC Maximum Performance Rate

	ASC 1X (ONE AU)		ASC 4X (FOUR AU'S)	
	32-BIT RESULTS/SEC	64-BIT RESULTS/SEC	32-BIT RESULTS/SEC	64-BIT RESULTS/SEC
ADD	16×10^6	9.2×19^6	64×10^6	37×10^6
MULTIPLY	16×10^6	5.3×10^6	64×10^6	21×10^6
DOT PRODUCT	16×10^6	4.0×10^6	64×10^6	16×10^6

It is a floating vector multiply instruction preceded by the loading of the vector parameter registers. Table II gives some typical code found in weather models. A double-nested DO LOOP with typical indexing conventions is shown in (1). (2) gives the sequence of instructions produced by the ASC compiler. All instructions are vectors, and the necessary indexing information for addressing purposes is contained in each vector parameter file. No scalar instructions are necessary in this example.

A powerful example of vector instruction capabilities is found in the use of the hardware-implemented dot-product operation. This operation consists of the multiplication of appropriate elements of two arrays followed by the sum of the products. To implement a matrix multiply operation from Fortran, the ASC compiler uses a single dot-product instruction and the complex indexing capability of the hardware to carry out the full matrix multiply. Three levels of addressing changes are implied in this case, and the hardware is designed to comprehend this level of indexing complexity.

The execution rate for the elementary operations of matrix multiply is one result per clock cycle for a one-pipe CP, or a rate of four results per clock cycle for a four-pipe CP. The compiler partitions the total matrix multiply across the appropriate number of pipes. Therefore, to complete a matrix multiply of two N by N matrices, a four-pipe CP will require approximately $N^3/4$ times the clock rate in seconds. This does not include the startup overhead necessary to fill the pipelines with operands.

TABLE IV—Relative Computer Capacity* Third Generation Systems

MFR	MODEL	RELATIVE SPEED
IBM	S/360 MODEL 65	1
IBM	S/360 MODEL 75	1.5
CDC	6500	1.5
CDC	6600	2.5
IBM	S/370 MODEL 165	3.5
IBM	S/360 MODEL 91	5
HITACHI	HITAC 8800	5
IBM	S/360 MODEL 95	7
CDC	7600	8
IBM	S/360 MODEL 195	8

* Data taken from Table E, page 546, Program for the study conference on the Modeling Aspects of GATE, Bulletin of the American Meteorological Society, Vol. 54 No. 6, June, 1973.

It is the authors' opinion that performance indices for array-oriented architectures are not meaningful when only the Millions of Instructions Per Second (MIPS) factor is used. Since a single vector instruction is equivalent to several scalar instructions (typically Load, Operation, Increment and Test Branch), and the number of data values used determines the number of execution of these scalar instructions, MIP ratings are ambiguous at best.

Consider the performance of an ASC producing "results per second." In this context "results per second" is the rate at which data fetched from central memory can be operated upon and the results stored back into central memory. Table III shows the maximum performance rates for one- and four-pipe ASC systems performing typical arithmetic operations. Assumptions are that the clock cycle is 60 nanoseconds and that the pipelines are already filled with operands. Vector dot product is a special case in the sense that the results per second rate pertains to the elementary operations.

Another performance measure can be determined from the present performance of ASC System #4 executing a particular weather benchmark. Although the benchmark is not a full weather prediction code, it does have the characteristic source code sequences and reflects the ability of the Fortran compiler to produce efficient code from a large applications package. Execution speed of the benchmark on the IBM Model 91 is approximately 246 minutes, and present ASC timing with checkout not finalized has already demonstrated approximately 30 minutes. This ratio of 8.2 is a measure of the total system performance upon this program. It reflects a mix of both scalar and vector instructions as well as I/O and other system services. The design of the ASC has been directed toward matching the real world mix of instructions encountered in typical applications instead of sacrificing scalar capability to provide vector capability.

In order to compare the observed ASC performance on the Weather Benchmark, data found in the Bulletin of the American Meteorological Society[1] is given in Table IV. Using the IBM S/360 Model 65 as the basis of reference, each of the systems listed is compared as to relative speed. Using the observed ASC/M91 ratio of 8.2, the present ASC speed would be 41 in the table.

ACKNOWLEDGMENTS

It would not be possible to acknowledge all the contributors to the development of the ASC; but particular recognition should be given to Messrs. H. G. Cragon, W. D. Kastner, E. H. Husband, D. R. Best, C. M. Stephenson, C. R. Hall, F. A. Galindo, E. C. Garth, and N. M. Chandler who contributed significantly to the development of the hardware. Software concepts are due in large part to the efforts of Messrs. L. C. Dean, G. T. Boswell, A. E. Riccomi, F. A. Little, W. Winkelman, W. L. Cohagan, and S. D. Nolte. Many other members of the Texas Instruments staff have also contributed immeasurably in the development of the ASC.

REFERENCES

1. Program for the study conference on the Modeling Aspects of Gate, *Bulletin of the American Meteorological Society*, Vol. 54, No. 6. June 1973, page 546, table E.

Multiprocessor performance analysis

by JOHN MITCHELL, CHARLES KNADLER, GARY LUNSFORD, and STEVE YANG

IBM Corporation
Morris Plains, New Jersey

INTRODUCTION

This paper reports the results of a study to determine the expected performance of a "classical" multiprocessor. The "classical" multiprocessor as used here is a multiprocessor consisting of K $(K>1)$ central processor units which have access to N storage units. The tool used for studying the performance is a discrete Fortran model of the hardware and software. Hardware measurements of the real system performance were available for validation of the model. The primary indicator of performance is the system thruput in millions of instructions executed per second (MIPS). Since the application of the hardware is in a real-time system, consideration is also given to response times of critical software tasks.

The multiprocessor hardware and software described in this paper were developed for a U.S. Government project (Contract Numbers DAHC-60-71-C-0005/0017) in which Western Electric Company and Bell Telephone Laboratories, Incorporated had prime responsibility. Sperry Univac was a major participant in the hardware development, together with Western Electric Company and Bell Telephone Laboratories, Incorporated. International Business Machines Corp. provided significant support to Bell Laboratories in software development.

HARDWARE DESCRIPTION

The multiprocessor of interest is shown in Figure 1. It is built up from Processor Units (PU), Program Stores (PS), Variable Stores (VS), and an Input/Output Controller (IOC). There are also spare units of each type that can be switched in for system recovery.

The Processor Unit (PU) contains all of the logic for executing the instruction set. It has direct access to Program Store for fetching instructions, and direct access to Variable Store for fetching instructions or fetching/storing data. Instruction length is 32-bit or 16-bit, depending on the format of the particular instruction. A fetch from PS is 64 bits (plus parity) or between two and four instructions. Fetches from Variable Store are normally 32-bit; two contiguous 32-bit words may be fetched with a single instruction. The processor unit consists of three subunits—Program Control (PCU), Operand Control (OCU), and Arithmetic Control (ACU). These three units operate asynchronously and in parallel.

Program Control Unit—The main function of the PCU is to fetch instructions from PS (or VS under certain conditions) and distribute these instructions to the other two control units. The PCU contains an input instruction buffer of four 64-bit words. This buffer tends to "insulate" the PCU from instantaneous contention at PS, since instructions can be fetched faster than the PU can execute them. The buffer is normally at least one word ahead of the ACU/OCU. On "branch" instructions the next instruction word from both sides of the branch is fetched. When the branch decision is made, one of these fetches is overwritten by the second word of the correct path. The PCU moves instructions from the buffer, partially translates, and passes the instructions to the ACU or OCU.

Operand Control Unit—The OCU controls the fetching/storing of operands to Variable Store. Most of its operation is controlled by instructions from the PCU.

Arithmetic Control Unit (ACU)—The ACU performs arithmetic or logical operations on data retrieved by the OCU or already in registers. The ACU has fixed point and floating point capability and all data is expressed in binary, two's complement, form.

The PU instruction execution time varies from 280 nanoseconds add to 5180 nanoseconds square root. The thruput is highly dependent on instruction mix and instruction sequence because of the high degree of parallelism in the execution logic.

A Program Store unit contains two modules of 8192 words (64 bits and 4 parity) each. The cycle time is nominally 500 nanoseconds for each module. The unit has two ports, one for each module, and the modules are two-way interleaved; i.e., odd addresses in one module, even addresses in the second module. If the interleaving were perfectly used, a PS could supply one word every 250 nanoseconds (4 megawords/second). With real code, the interleaving is never fully utilized and the supply rate is in the range of three to four megawords/second.

Words in PS cannot be modified by the PU; i.e., it is a read only store in the normal configuration. PS is loaded via the IOC from tape or disk under control of the operating system.

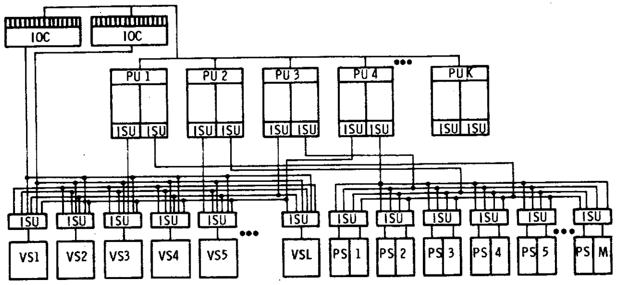


Figure 1—Multiprocessor configuration

A Variable Store unit contains one module of 16,384 64-bit words. The cycle time is nominally 500 nanoseconds and either one 32-bit word or two contiguous 32-bit words can be fetched in one cycle time. Variable store is normally used for variables but it can also be addressed for instructions. The instruction mode is used for routines which are rolled in at very low data rates, for example display formats.

The IOC is a programmable unit which controls all I/O operations. It has direct access to VS and operates on a non-interference basis except when there is contention by PU's at a VS. It provides sixteen channels, with lower numbered channels serviced more often than higher numbered channels.

SOFTWARE DESCRIPTION

The entire collection of real-time software for execution on one multiprocessor is called a process. The process is composed of timed arrays, where one timed array accomplishes some defined function. The timed array is composed of one or more tasks which in turn are composed of routines and subroutines. The task is the basic software unit and is a single-process entity; i.e., it is not multiprocessed. In a seven PU system, seven tasks can be executing concurrently. These seven tasks may belong to the same timed array or to several different timed arrays. The routines and subroutines that make up the task may be located in different program stores. In fact, one result of the study indicated the desirability of locating the subroutines of a task in different program stores to maximize system thruput.

Tasks become eligible for execution through enablement. There may be several events required before a task is enabled; e.g., predecessor tasks are completed or an I/O operation is completed. When all of the predecessor conditions are satisfied the task is "absolutely enabled" and is placed on a queue for execution. Associated with each task is a preassigned priority number, based on the importance of the task to accomplishing the mission of the system. Tasks are placed on the queue in order of their priority number. When a processor is available, the highest priority absolutely enabled task will execute on the processor.

A module of the Operating System, the dispatcher, manages the assignment of tasks to processors. The dispatcher executes on any processor which is available. Its job is to search the queue of tasks, find the highest priority task which is enabled and transfer control of the processor to that task. Notice that under light load, several processors may be executing the dispatcher simultaneously, looking for a task to execute. Once a task begins execution, it will run to completion on its processor unless an error condition occurs.

MODEL DESCRIPTION

A Fortran model of the hardware/software system was developed for use as an analysis tool. The most significant design problem of the model was development of an algorithm to simulate the effects of contention at the memory units. The algorithm which was developed gives good results, as determined by comparison of its output with hardware measurements.

The contention algorithm uses three parameters: access ratio, memory supply rate, and processor demand. The access ratio (AR) is defined as:

$$AR = \frac{\text{Number of instructions executed}}{\text{Number of words (64 bit) fetched}}$$

and is characteristic of each software routine. Recall that the

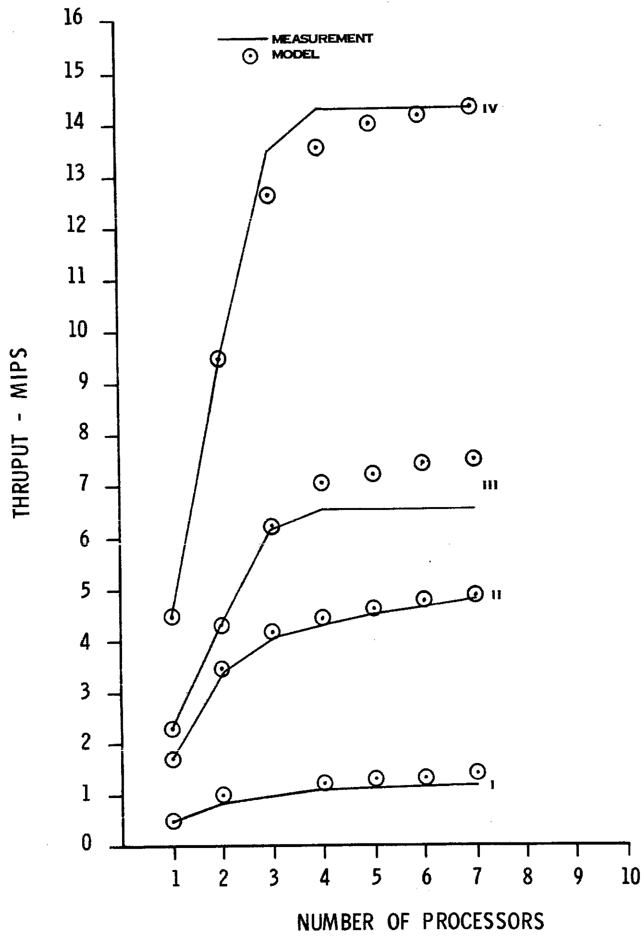


Figure 2—Speed curves, 1 program store

instruction length may be 16 bit or 32 bit and also that both paths of a branch instruction are fetched. Consequently the AR varies from 2 to 4 if there are no branch instructions, and may be as low as 0.3 if there are many branches. Analysis of the applications code indicated that AR varied from 0.6 to 2.2 with an average of 1.4.

The memory supply rate (K) is the words per second that a memory can supply under various conditions. The variable store unit (VS) is not interleaved and can supply words at a maximum rate of 1/access time or 2 words/USEC. The program store unit (PS) has two ports and is two-way interleaved; i.e., even addresses in one module, odd addresses in the other module. For PS, the supply rate is a function of the number of processors accessing it, and the time distribution of the access requests. Consider two processor units using one PS. We assume that consecutive requests from each PU alternate from odd address to even address and that the requests are randomly distributed in time. Then on the average, both PU's will need a word from the same module one-half the time and from different modules one-half the time. Under these conditions, the supply rate is 3 words/USEC (K_2). Similar reasoning for three PU's accessing one PS gives $K_3 = 3.5$ words/USEC, etc.

The processor demand Rpu is the number of instructions needed by the processor during each time step of the simulation and is derived from the number of instructions in the routine being executed.

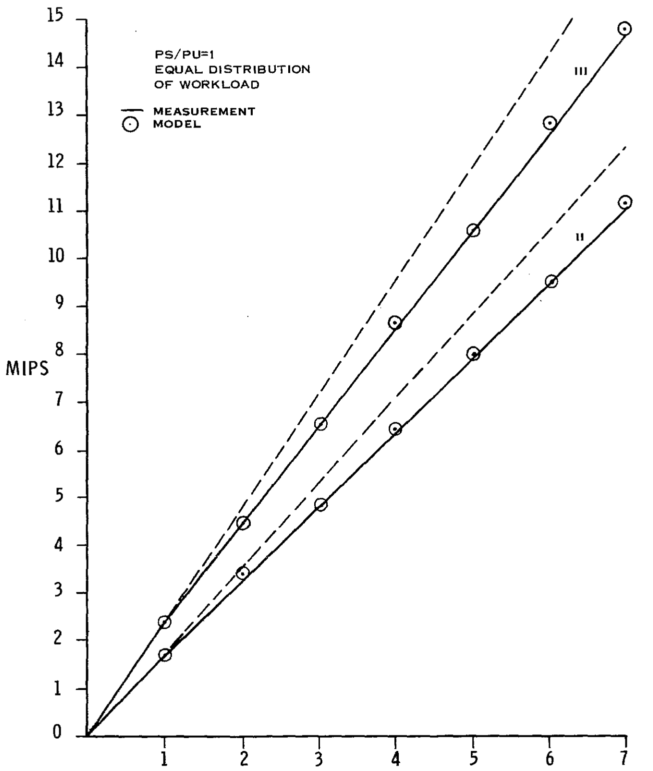


Figure 3—Throughput curves

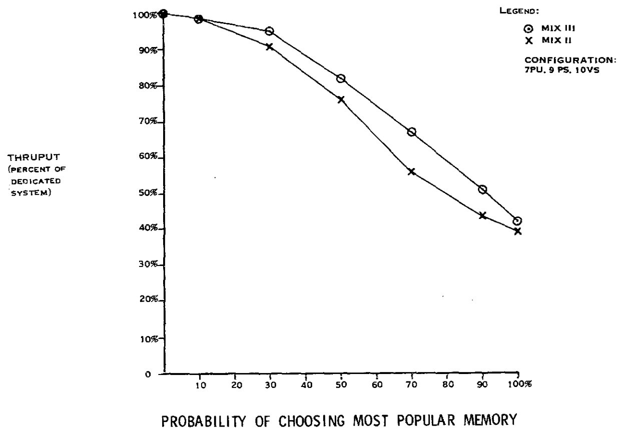


Figure 4—Sensitivity of thruput to biased PS memory usage

The contention algorithm used is:

For each PU(PU_n) at each Δt:

(1) Determine the PU demand, R_{pu_n}, and the PS number
(2) Determine the total number of PU's (k) accessing the PS number determined in Step 1.
(3) $R^*_{pu_n} = \min\left(R_{pu_n}, \frac{R_{pu_n}}{\sum_{1}^{k} Rpu_l} K_k \times AR\right)$
(4) Repeat 1, 2, 3 for VS to obtain $R_{pu_n}{}^{**}$
(5) $R_{pu_n}{}^{***} = \min(R_{pu_n}{}^{*}, R_{pu_n}{}^{**})$
(6) Performance Ratio (PR) $= R_{pu_n}{}^{***}/R_{pu_n}$

The performance ratio is then used in the model to determine the number of instructions actually processed during the time step Δt.

MODEL FEATURES

The driver for the model is a time line of task enablements which is derived from a scenario to which the real system must respond. The modules of the operating system which are concerned with task management and dispatching are modelled such that they are functionally identical to the real system. Applications code is characterized by priority, number of instructions, access ratio, VS fetches, and PS assignment.

An extensive output module was designed so that a wide range of problems could be investigated. For example, a time profile of storage utilization can be obtained, which can be used to relocate programs in PS to minimize instances of high PS utilization. Also data concerning task queueing times are available, which are used to study response time.

STUDY RESULTS

The performance of the data processor, using the model, was studied in three phases. The first phase was to duplicate

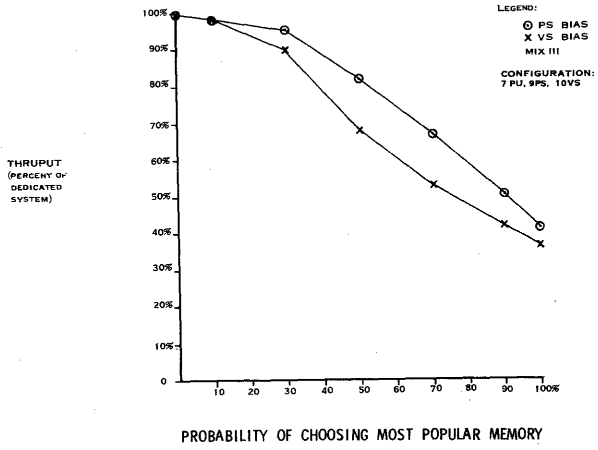


Figure 5—Sensitivity of thruput to biased memory usage

some experiments performed on the real system, in order to validate the hardware section of the model. The second phase was a series of sensitivity analyses to determine the parameters which had the greatest effect on system thruput. The third phase was an evaluation of system performance using a detailed characterization of the application software.

Validation experiments

Validation experiments were conducted using four different instruction mixes, and the results compared with hardware measurements. Two types of curves were obtained: speed curves, wherein the number of processors was varied from one to seven, with only one program store; thruput curves, where the number of program stores equals the number of processors. The characteristics of the four mixes are shown below:

Mix	Uniprocessor Thruput (MIPS)	Access Ratio
I	0.50	.34
II	1.76	1.24
III	2.41	1.98
IV	4.72	3.65

Mix I consists of all branch type instructions and is a "worst case" mix. Mix II and III are arithmetic and logical instruction mixes and are representative of the applications code. Mix IV consists of all "No Op" instructions and is a maximum thruput mix.

The speed curves for these four mixes obtained from the model and from hardware measurement are shown in Figure 2. Good agreement between the model and measurement was obtained for Mix I and Mix II. Mix III is characterized by an imbalance in the number of words fetched from each module of the program store. Thus one of the assumptions of the model, "perfect interleaving," is violated. Mix IV tends to be very time synchronous in its execution since all instructions are the same. This results in less contention at the program store than predicted by the model.

The thruput curves for Mixes II and III are shown in Figure 3. For these curves, the number of program stores is set equal to the number of processors. Each time a processor completes execution, the program store from which the next execution is to be made is randomly selected from all program stores with equal probability. The dashed lines are straight line extrapolations of the uniprocessor thruput. Notice that if the applications program are designed such that an equal distribution of PS usage obtains, there is very little reduction in thruput due to PS contention.

Sensitivity analyses

A series of studies were undertaken to determine the sensitivity of the system thruput to changes in several software parameters. The parameters considered were access ratio (ratio of instructions executed to program store words fetched) and bias in memory usage.

For a configuration consisting of seven PU's, nine PS's, and ten VS's and a software design that equally distributes demand for memories, the probability that a processor will use a particular memory is 1/10 for VS and 1/9 for PS. In performing the memory bias studies, the VS and PS probabilities were varied to produce results which represented memory biases ranging from the dedicated use of memories to all processors using the same memory.

Figure 4 presents the sensitivity dependence of thruput upon Program Store utilization bias with a seven PU, nine PS and ten VS configuration for the two representative instruction mixes II and III. The ordinate of the graph represents the achieved thruput as a percentage of dedicated thruput; i.e., the thruput achieveable when each PU executes out of its own dedicated PS. The abscissa gives the percentage probability of choosing the most popular PS (PS #1) where a probability of 1.0 equals 100 percent. The zero

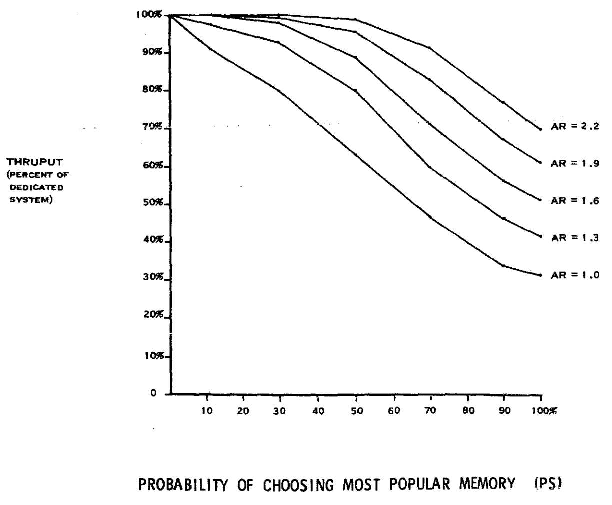


Figure 6—Sensitivity of thruput to biased memory usage, 7PU 9PS

probability case is by definition the dedicated case. The data were obtained under conditions of zero VS queuing.

It can be seen from Figure 4 that for a bias of 50 percent the thruput falls to 75.8 percent of dedicated thruput for mix II and 83 percent of dedicated thruput for mix III, or mip rates of 9.35 and 13.88 respectively. These results indicate that even with substantial bias (50 percent) in PS #1 usage, the system provides thruput of greater than 1.3 mips/PU for two representative instruction mixes.

Figure 5 indicates the relationship between thruput sensitivity and Variable Store Utilization bias for the seven PU, nine PS, ten VS configuration with equal probability of addressing all PS's. The ordinate of the graph is the achieved thruput as a percentage of the dedicated thruput. The abscissa gives the probability of choosing the most popular memory (VS #1). The zero probability case is by definition the dedicated case. The PS bias data are also given for ease of comparison.

The ratio of VS accesses to PS accesses is 0.642 for mix III, which is considered to be significantly greater than the ratio of 0.40 found in the application programs. Thus, the effects of VS queuing are magnified in Figure 5 and one observes for this mix that VS bias has greater effect upon thruput than an equivalent bias in PS usage. For example, a 50 percent bias in VS usage causes a drop in thruput to 68 percent, while a 50 percent bias in PS usage reduces thruput to 82 percent.

The results of the Access Ratio Sensitivity study are presented in Figure 6. The data for Figure 6 were obtained using the characteristics of mix II and by varying access ratio. The ordinate of the Figure is the thruput expressed as a percentage of that which a dedicated system can achieve. The abscissa is access ratio while the family of curves represents different PS usage biases. The results in Figure 6 indicate that thruput is very sensitive to coupled variations in PS bias and access ratio. However, in the range of 0 percent to 50 percent PS usage bias, Figure 7 shows that there is less thruput dependence upon access ratio. Figure 7 presents the same basic data as Figure 6, except the abscissa is now PS usage bias. A parametric family of curves is generated with a unique thruput curve for each access ratio. Figure 7 depicts a relatively slow fall-off in thruput with PS bias up to 50 percent for access ratios greater than 1.3.

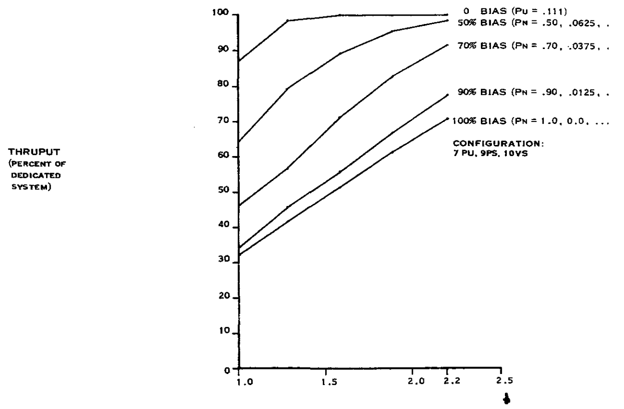


Figure 7—Sensitivity of thruput to access ratio

System studies

The model was used to investigate the system thruput performance when executing the application programs. All of the programs were characterized by their priority, number of instructions, access ratio, and variable store accesses, and were run under a simulation of the operating system. Many model runs were made under different loads and system configurations; three runs of interest are presented here. The conditions for these runs were:

Run 1.	All code for each program assigned to a single PS. VS accesses not simulated. Light load.
Run 2.	Same as Run 1 except heavy load.
Run 3.	Code for each program distributed across several PS. VS accesses included. Heavy load.

The results of these runs are shown in Table I. We observe that the overhead (dispatcher) is reasonably constant, independent of system load. Also there was a significant increase in system thruput when the tasks were distributed in Run 3.

TABLE I—System Results

	System Thruput	% Utilization		
Run Number	MIPS (7 pu)	Applications	Dispatcher	Idle
1	8.49	34	14	52
2	9.30	72	14	14
3	10.89	80	13	6

CONCLUSIONS

The model that was developed is a useful tool for investigating the system performance under various loads. In comparison with measurements on the real system, the model provided reasonably close results. Of more general interest is the conclusion that the performance of a multiprocessor of the type described here can be optimized by a software system design which achieves near uniform distribution of the storage unit usage.

ACKNOWLEDGMENTS

The authors would like to thank Mr. D. C. Ruberg of Bell Telephone Laboratories, Inc., Whippany, N.J. for providing data on the performance of the hardware.

STARAN parallel processor system hardware

by KENNETH E. BATCHER

Goodyear Aerospace Corporation
Akron, Ohio

INTRODUCTION

The parallel processing capability of STARAN* resides in n array modules ($n \leq 32$). Each array module contains 256 small processing elements (PE's). They communicate with a multi-dimensional access (MDA) memory through a "flip" network, which can permute a set of operands to allow inter-PE communication. This gives the programmer a great deal of freedom in using the processing capability of the PE's. At one stage of a program, he may apply this capability to many bits of one or a few items of data; at another stage, he may apply it to one or a few bits of many items of data.

The remainder of this paper deals with the MDA memories, the STARAN array modules, the other elements of STARAN, and the results of certain application studies.

MULTI-DIMENSIONAL ACCESS (MDA) MEMORIES

A common implementation of associative processing is to treat data in a bit-sequential manner. A small one-bit PE (processing element) is associated with each item or word of data in the store, and the set of PE's accesses the data store in bit-slices; a typical operation is to read Bit i of each data word into its associated PE or to write Bit i from its associated PE.

The memory for such an associative processor could be a simple random-access memory with the data rotated 90 deg, so that it is accessed by bit-slices instead of by words. Unfortunately, in most applications, data come in and leave the processor as items or words instead of as bit-slices. Hence, rotating the data in a random-access memory complicates data input and output.

To accommodate both bit-slice accesses for associative processing and word-slice accesses for STARAN input/output (I/O), the data are stored in a multi-dimensional access (MDA) memory (Figure 1). It has wide read and write busses for parallel access to a large number (256) of memory bits. The write mask bus allows selective writing of memory bits. Memory accesses (both read and write accesses) are controlled by the address and access mode control inputs; the access mode selects a stencil pattern of 256 bits, while the address positions the stencil in memory.

For many applications, the MDA memory is treated as a square array of bits, 256 words with 256 bits in each word. The bit-slice access mode (Figure 2A) is used in the associative operations to access one bit of all words in parallel, while the word access mode (Figure 2B) is used in the I/O operations to access several or all bits of one word in parallel.

The MDA memory structure is not limited to a square array of 256 by 256. For example, the data may be formatted as records with 256 8-bit bytes in each record. Thirty-two such records can be stored in an MDA memory and accessed several ways. To input and output records, one can access 32 consecutive bytes of a record in parallel (Figure 3A). To search key fields of the data, one can access the corresponding bytes of all records in parallel (Figure 3B). To search a whole record for the presence of a particular byte, one can access a bit from each byte in parallel (Figure 3C).

The MDA memories in the STARAN array modules are bipolar. They exhibit read cycle times of less than 150 nsec and write cycle times of less than 250 nsec.

STARAN ARRAY MODULES

A STARAN array module (Figure 4) contains an MDA memory communicating with three 256-bit registers (M, X, and Y) through a flip (permutation) network. One may think of an array module as having 256 small processing elements (PE's), where a PE contains one bit of the M register, one bit of the X register, and one bit of the Y register.

The M register drives the write mask bus of the MDA memory to select which of the MDA memory bits are modified in a masked-write operation. The MDA memory also has an unmasked-write operation that ignores M and modifies all 256 accessed bits. The M register can be loaded from the other components of the array module.

In general, the logic associated with the X register can perform any of the 16 Boolean functions of two variables;

* TM, Goodyear Aerospace Corporation, Akron, Ohio.

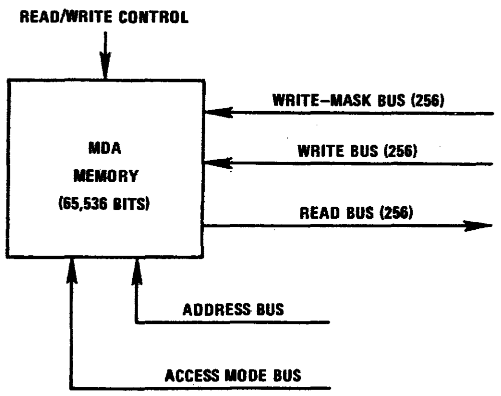


Figure 1—Multi-dimensional access memory

that is, if x_i is the state of the ith X-register bit, and f_i is the state of the ith flip network output, then:

$$x_i \leftarrow \phi(x_i, f_i) \quad (i=0, 1, \ldots, 255)$$

where ϕ is any Boolean function of two variables. Similarly, the logic associated with the Y-register can perform any Boolean function:

$$y_i \leftarrow \phi\ (y_i, f_i) \quad (i=0, 1, \ldots, 255)$$

where y_i is the state of the ith Y-register bit. The programmer is given the choice of operating X alone, Y alone, or X and Y together.

If X and Y are operated together, the same Boolean function, ϕ, is appled to both registers:

$$x_i \leftarrow \phi\ (x_i, f_i)$$
$$y_i \leftarrow \phi\ (y_i, f_i)$$

The programmer also can choose to operate on X selectively, using Y as a mask:

$$x_i \leftarrow \phi\ (x_i, f_i) \quad \text{(where } y_i=1)$$
$$x_i \leftarrow x_i \quad \text{(where } y_i=0)$$

Another choice is to operate on X selectively while operating on Y:

$$x_i \leftarrow \phi\ (x_i, f_i) \quad \text{(where } y_i=1)$$
$$x_i \leftarrow x_i \quad \text{(where } y_i=0)$$
$$y_i \leftarrow \phi\ (y_i, f_i)$$

In this case, the old state of Y (before modification by ϕ) is used as the mask for the X operation.

For a programming example, the basic loop of an unmasked add fields operation is selected. This operation adds the contents of a Field A of all memory words to the contents of a Field B of the words and stores the sum in a Field S of the words. For n-bit fields, the operation executes the basic loop n times. During each execution of the loop, a bit-slice (a) of Field A is ready from memory, a bit-slice (b) of Field B is read, and a bit-slice (s) of Field S is written into memory. The operation starts at the least significant bits of the fields and steps through the fields to the most significant bits. At the beginning of each loop execution, the carry (c) from the previous bits is stored in Y, and X contains zeroes:

$$x_i=0$$
$$y_i=c_i$$

The loop has four steps:

Step 1: Read Bit-slice a and exclusive-or ($\oplus$) it to X selec-

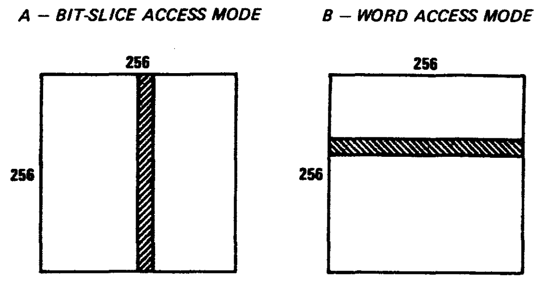


Figure 2—Bit-slice and word access modes

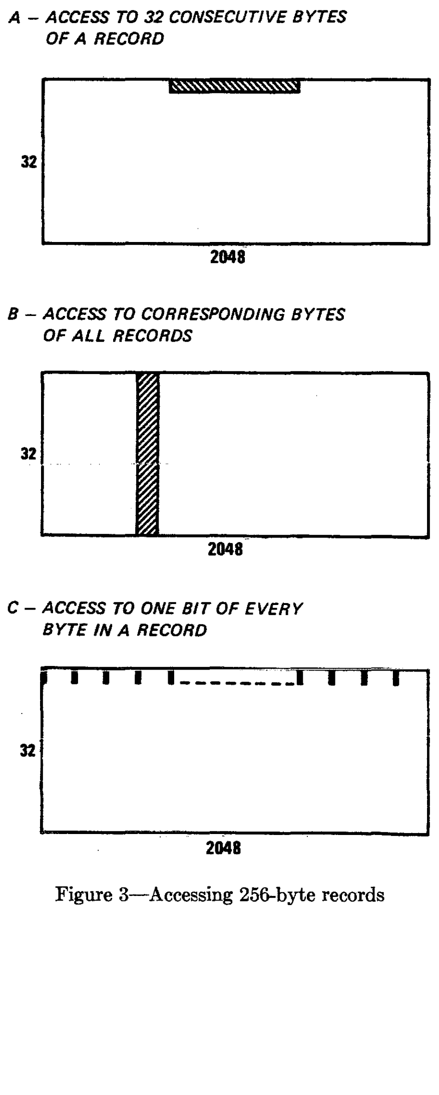

A – ACCESS TO 32 CONSECUTIVE BYTES OF A RECORD
32
2048
B – ACCESS TO CORRESPONDING BYTES OF ALL RECORDS
32
2048
C – ACCESS TO ONE BIT OF EVERY BYTE IN A RECORD
32
2048

Figure 3—Accessing 256-byte records

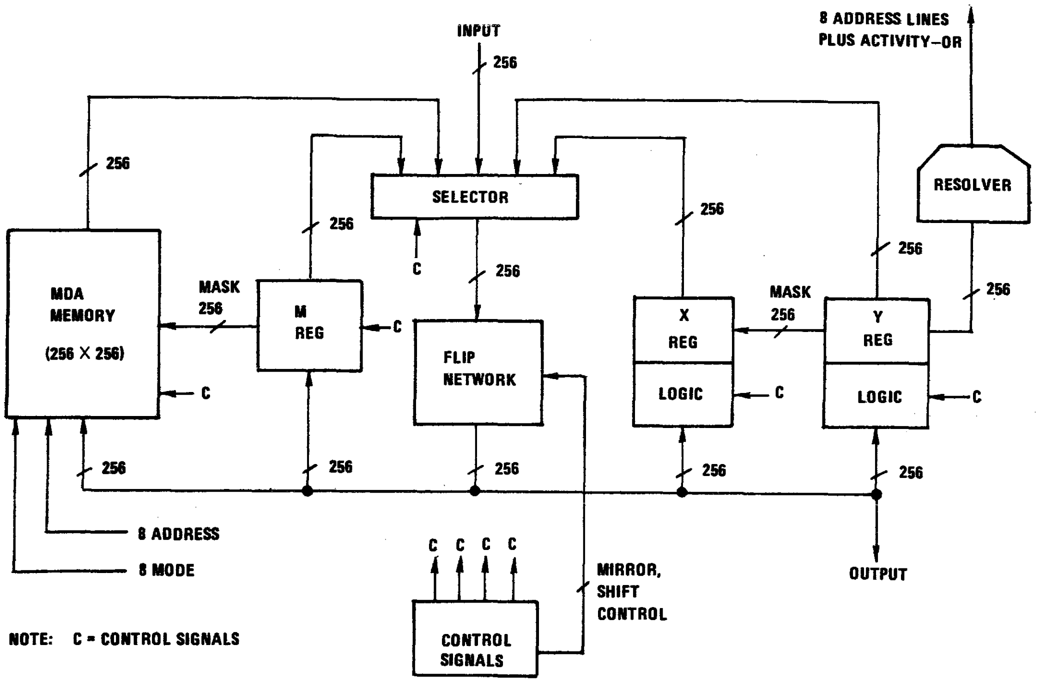


Figure 4—STARAN array module

tively and also to Y:

$$x_i \leftarrow x_i \oplus y_i a_i$$

$$y_i \leftarrow y_i \oplus a_i$$

The states of X and Y are now:

$$x_i = a_i c_i$$

$$y_i = a_i \oplus c_i$$

Step 2: Read Bit-slice *b* and exclusive-or it to X selectively and also to Y:

$$x_i \leftarrow x_i \oplus y_i b_i$$

$$y_i \leftarrow y_i \oplus b_i$$

Registers X and Y now contain the carry and sum bits:

$$x_i = a_i c_i \oplus a_i b_i \oplus b_i c_i = c'_i$$

$$y_i = a_i \oplus b_i \oplus c_i = s_i$$

Step 3: Write the sum bit from Y into Bit-slice *s* and also complement X selectively:

$$s_i \leftarrow y_i$$

$$x_i \leftarrow x_i \oplus y_i$$

The states of X and Y are now:

$$x_i = c'_i \oplus s_i$$

$$y_i = s_i$$

Step 4: Read the X-register and exclusive-or it into both X and Y:

$$x_i \leftarrow x_i \oplus x_i$$

$$y_i \leftarrow y_i \oplus x_i$$

This clears X and stores the carry bit into Y to prepare the registers for the next execution of the loop:

$$x_i = 0$$

$$y_i = c'_i$$

Step 3 takes less than 250 nsec, while Steps 1, 2, and 4 each take less than 150 nsec. Hence, the time to execute the basic loop once is less than 700 nsec. If the field length is 32 bits, the add operation takes less than 22.4 microsec plus a small amount of setup time. The operation performs 256 additions in each array module. This amounts to 1024 additions, if four array modules are enabled, to achieve a processing power of approximately 40 MIPS (million-instructions-per-second).

The array module components communicate through a network called the flip network. A selector chooses a 256-bit source item from the MDA memory read bus, the M register, the X register, the Y register, or an outside source. The bits of the source item travel through the flip network, which may shift and permute the bits in various ways. The permuted source item is presented to the MDA memory write bus, M register, X register, Y register, and an outside destination.

The permutations of the flip network allow inter-PE communication, A PE can read data from another PE either directly from its registers or indirectly from the MDA memory. One can permute the 256-bit data item as a whole or divide it into groups of 2, 4, 8, 16, 32, 64, or 128 bits and permute within groups.

The permutations allowed include shifts of 1, 2, 4, 8, 16, 32, 64, or 128 places. One also can mirror the bits of a group (invert the left-right order) while shifting it. A positive shift of mirrored data is equivalent to a negative shift of the unmirrored data. To shift data a number of places, multiple passes through the flip network may be required. Mirroring can be used to reduce the number of passes. For example, a shift of 31 places can be done in two passes: mirror and shift 1 place on the first pass, and then remirror and shift 32 places on the second pass.

The flip network permutations are particularly useful for fast-fourier transforms (FFT's). A 2^n point FFT requires n steps, where each step pairs the 2^n points in a certain way and operates on the two points of each pair arithmetically to form two new points. The flip network can be used to rearrange the pairings between steps. Bitonic sorting (2) and other algorithms (3) also find the permutations of the flip network useful.

Each array module contains a resolver reading the state of the Y register. One output of the resolver (activity-or) indicates if any Y bit is set. If some Y bits are set, the other output of the resolver indicates the index (address) of the first such bit. Since the result of an associative search is marked in the Y register, the resolver indicates which if any words respond to the search.

OTHER STARAN ELEMENTS

Figure 5 is a block diagram of a typical STARAN system with four array modules. Each array module contains an assignment switch that connects its control inputs and data inputs and outputs to AP (associative processor) control or the PIO (parallel input/output) module.

The AP control unit contains the registers and logic necessary to exercise control over the array modules assigned to it. It receives instructions from the control memory and can transfer 32-bit data items to and from the control memory. Data busses communicate with the assigned array modules. The busses connect only to 32-bits of the 256-bit-wide input and output ports of the array modules (Figure 4), but the permutations of the array module flip networks allow communication with any part of the array. The AP control sends control signals and MDA memory addresses and access modes to the array modules and receives the resolver outputs from the array modules.

Registers in the AP control include:

1. An instruction register to hold the 32-bit instruction being executed.
2. A program status word to hold the control memory address of the next instruction to be executed and the program priority level.
3. A common register to hold a 32-bit search comparand, an operand to be broadcast to the array modules, or an operand output from an array module.
4. An array select register to select a subset of the assigned array modules to be operated on.
5. Four field pointers to hold MDA memory addresses and allow them to be incremented or decremented for stepping through the bit-slices of a field, the words of a group, etc.
6. Three counters to keep track of the number of executions of loops, etc.
7. A data pointer to allow stepping through a set of operands in control memory.
8. Two access mode registers to hold the MDA memory access modes.

The parallel input/output (PIO) module contains a PIO flip network and PIO control unit (Figure 5). It is used for high bandwidth I/O and inter-array transfers.

The PIO flip network permutes data between eight 256-bit ports. Ports 0 through 3 connect to the four array modules through buffer registers. Port 7 connects to a 32-bit data bus in the PIO control through a fan-in, fan-out switch. Ports 4, 5, and 6 are spare ports for connections to high bandwidth peripherals, such as parallel-head disk stores, sophisticated displays, and radar video channels. The spare ports also could be used to handle additional array modules. High bandwidth inter-array data transfers up to 1024 bits in parallel are handled by permuting data between Ports 0, 1, 2 and 3. Array I/O is handled by permuting data between an array module port and an I/O port. The PIO flip network is controlled by the PIO control unit.

The PIO control unit controls the PIO flip network and the array modules assigned to it. While AP control is processing data in some array modules the PIO control can input and output data in the other array modules. Since most of the registers in the AP control are duplicated in PIO control, it can address the array modules associatively.

The control memory holds AP control programs, PIO control programs, and microprogram subroutines. To satisfy the high instruction fetch rate of the control units (up to 7.7 million instructions per second), the control

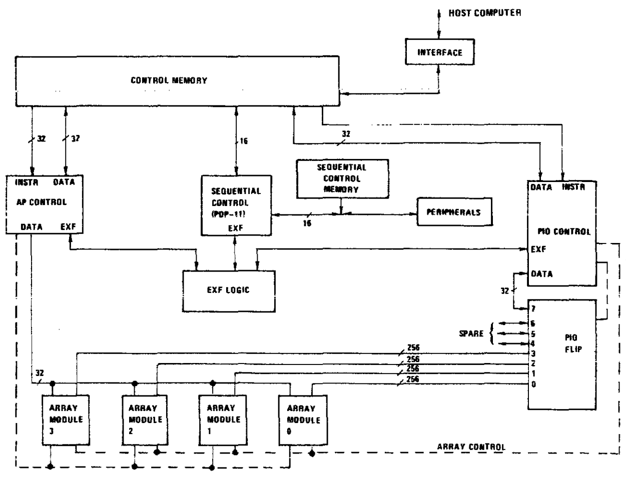


Figure 5—Typical STARAN block diagram

memory has five banks of bipolar memory with 512 32-bit words in each bank. Each bank is expandable to 1024 words. To allow for storage of large programs, the control memory also has a 16K-word core memory with a cycle time of 1 microsec. The core memory can be expanded to 32K words. Usually the main program resides in the core memory, and the system microprogram subroutines reside in bipolar storage. For flexibility, users are given the option of changing the storage allocation and dynamically paging parts of the program into bipolar storage.

A Digital Equipment Corporation (DEC) PDP-11 minicomputer is included to handle the peripherals, control the system from console commands, and perform diagnostic functions. It is called sequential control to differentiate it from the STARAN parallel processing control units. The sequential control memory of 16K 16-bit words is augmented by a 8K×16-bit "window" into the main control memory. By moving the window, sequential control can access any part of control memory. The window is moved by changing the contents of an addressable register.

The STARAN peripherals include a disk, card reader, line printer, paper-tape reader/punch, console typewriter, and a graphics console.

Synchronization of the three control units (AP control, sequential control, and PIO control) is maintained by the external function (EXF) logic. Control units issue commands to the EXF logic to cause system actions and read system states. Some of the system actions are: AP control start/stop/reset, PIO control start/stop/reset, AP control interrupts, sequential control interrupts, and array module assignment.

The design of STARAN allows it to be connected to other computers (host computers) as a special-purpose peripheral. The interface can take many different forms. One could connect to an I/O channel of the host. Alternately, one could connect to the memory bus of the host so that it can address STARAN memory directly and/or allow STARAN to address its memory directly. For example, the STARAN at Rome Air Development Center (4) is connected to an I/O channel of a Honeywell HIS-645 computer. At Goodyear Aerospace, another STARAN is interfaced to the direct memory access port of an XDS Σ 5 computer.

APPLICATIONS

Several representative application areas—fast Fourier transforming, sonar post-processing, string search, file processing, and air traffic control—are discussed below. Other application-oriented work which has been performed under contract to various government agencies, include image processing, data management, position locating and reporting, bulk filtering and radar processing.

Fast fourier transform

The Fast Fourier Transform (FFT) is a basic operation in digital signal processing which is being widely used in the real-time processing of radar and sonar signals. The structure of the FFT algorithm is such that it can be segmented into many similar concurrent operations. Parallel implementation of the FFT can provide orders of magnitude speed increases over sequential computer execution times. The organization of STARAN lends itself to efficient manipulation of data in the FFT.

The Air Force supplied real radar data (on tapes) to GAC to be transformed by the STARAN system. A 512-point, 16-bit FFT was performed on this real data in 2.7 milliseconds using only two MDA arrays. A 1024-point transform on real input data could be performed in about 3.0 milliseconds using all four arrays available at GAC's STARAN evaluation and training facility. For comparison purposes, the following is a list of reported execution times for a 1024-point, real input, FFT:

Sequential Computers	
XDS Sigma 5	660 msec
IBM 360/67	446 msec
UNIVAC 1108	190 msec (complex)
UNIVAC 1108 (with array processor attachment)	29.2 msec (complex)
Special Purpose FFT Systems	
Time/Data 90 System	28 msec
ELSYTEC 306/HFFT	18 msec
SPECTRA SYSTEM '900'	9.2 msec

Sonar post-processing

Sensor data processing can be split into two major categories—signal processing and post-processing. Signal processing is the area of the system where operations such as the FFT are performed; post-processing involves the sorting and editing of the signal processor output data to determine tactical information such as whether a real target is in the coverage area and where the target is.

The job of sorting the spectral lines that result from the FFT operations is a formidable task, especially in a multisensor case. The trend has been for increasing the sensitivity of signal processing systems. The acoustic signal line sorting task that accompanies any increased sensitivity can be staggering. For instance, a 6 db improvement in sensitivity, in a classified Navy sonar system, would result in increasing the target load by a factor of 16 and the computer processing load by a factor of 250 or more.

A digital sonar signal processing system, under development at the Naval Air Development Center (NADC), requires that subroutines operate on the target spectral lines (outputs from an FFT) and other input data to form outputs suitable for later use in classification algorithms. Since the system is a multi-sensor system, these subroutines must process a very large volume of data in real-time. The content addressability feature of STARAN provides the potential for significant performance gains due to the

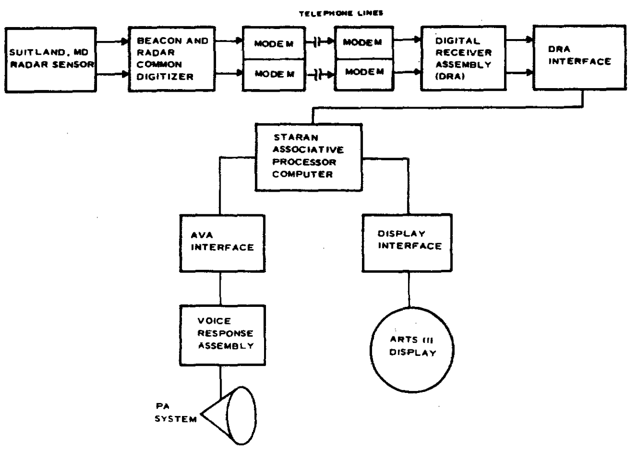

TELEPHONE LINES
SUITLAND, MD RADAR SENSOR
BEACON AND RADAR COMMON DIGITIZER
MODEM
MODEM
MODEM
MODEM
DIGITAL RECEIVER ASSEMBLY (DRA)
DRA INTERFACE
STARAN ASSOCIATIVE PROCESSOR COMPUTER
AVA INTERFACE
DISPLAY INTERFACE
VOICE RESPONSE ASSEMBLY
ARTS III DISPLAY
PA SYSTEM

Figure 6—TRANSPO '72 demonstration system

requirement for many searches in these post-processing subroutines.

As a consequence of this potential improvement, NADC issued a contract to GAC to assess the comparative run times for the STARAN versus a large-scale conventional computer (the CDC-6600). NADC-developed algorithms for the most time consuming operations in the post-processor system were programmed on the STARAN computer. Real data was then processed on both the STARAN and, by NADC, on a CDC-6600.

The STARAN executed the programs, using the real data, 200 times faster than the CDC-6600.

String search

A processing function used by several agencies for locating specific character strings (such as place names) in textual information, was developed for STARAN and tested on a sample data base. The same function was executed on a conventional computer (Sigma 5) for a timing comparison. The STARAN solution ran 100 times faster. This function is also applicable to nondefense applications such as patent, legal, and chemical information searches where cost of search may be a limiting parameter.

File processing

A personnel record file was used as a sample data base for demonstrating multiple-key searches. STARAN and a parallel-head disk were used for the demonstration. This work demonstrated that a query, simple or complex, can be processed in less than 120 milliseconds and that several queries may be batched and processed in the same processing time period. The simplicity of the software for retrieval and update was also demonstrated.

Air traffic control

In May 1972 a complete terminal automation system was demonstrated at the TRANSPO '72 exhibit at Dulles International Airport and later at private showings in Washington and Boston. Live radar and beacon sensor data were provided from the FAA site at Suitland, Maryland. The complete system is shown in Figure 6. The following ATC functions were demonstrated: beacon tracking, radar tracking, conflict prediction, conflict resolution, display processing, automatic voice advisory (AVA), and terrain avoidance.

The TRANSPO demonstration illustrated the use of STARAN in a full repertoire of terminal automation functions including advanced features such as automatic track initiation of all the aircraft, automatic tag placement on the display, and automatic handoff from sector to sector. The live targets were supplemented with 256 simulated targets so that up to 400 targets (representative of larger terminal densities) could be provided.

Average execution times for the most important functions were:

conflict prediction	90 msec
conflict resolution	25 msec
tracking	100 msec
display processing	160 msec

The entire ATC program executed in less than 5 percent of real-time.

REFERENCES

1. Batcher, K. E., "Flexible Parallel Processing and STARAN," *1972 WESCON Technical Papers*, Session 1.
2. Batcher, K. E., "Sorting Networks and Their Applications," 1968 Spring Joint Computer Conference, *AFIPS Proceedings*, Vol. 32, pp. 307-314.
3. Stone, H. S., "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers*, Vol. C-20, No. 2, February 1971, pp. 153-161.
4. Feldman, J. D., *RADCAP: An Operational Parallel Processing Facility*, Goodyear Aerospace Corporation.

A program for software quality control

by PAUL OLIVER

Department of the Navy
Washington, D.C.

INTRODUCTION

The Automatic Data Processing Equipment Selection Office (ADPESO) of the Department of the Navy is engaged in a development program for software to be used in the quantification of computer systems selection criteria, and the application of quality control procedures to selected software products.

That such a program be undertaken by this centralized ADPE Selection Office is both proper and important to the successful performance of our mission. This mission, briefly stated, is to evaluate and select, or review the selection of, commercially available automatic data processing equipment for approval by the Assistant Secretary of the Navy for Financial Management.

This development program is the responsibility of ADPESO's Software Development Division, and is concentrated in three areas. A COBOL Compiler Validation System has been designed, implemented, and is being used throughout the Federal Government to determine the degree to which COBOL compilers conform to the published standard. A system to facilitate the process of COBOL benchmark program conversion, evaluation, and implementation has been completed and is being field tested. Finally, an experiment in using a library of synthetic programs for system performance measurement is being conducted.

An evaluation of such a program requires a description of the projects, the identification of project controls which have been applied, and the resultant or expected payoffs. These will be discussed in turn.

Why a quality control program?

The problem we are attempting to alleviate is a financial one. During fiscal year 1973 ADPESO participated in 189 acquisition actions with a monthly rental value of $691,000. This does not include 173 reutilization actions. The scope of these actions is quite broad. Recent acquisitions have included 100 mini-computer configurations, 50 key-to-disk configurations, and a medical laboratory information system.

How do the above dollar figures relate to software? Precise measurements are difficult, but we estimate that the Department of the Navy's annual software expenditure is approximately three times that of hardware. Barry Boehm has cited a similar figure for the U. S. Air Force,[1] and we suspect this figure is fairly universal.

Our present work has as one of its principal purposes the lowering of software production and maintenance costs. These costs will of course vary with the nature of the system in question. A 1964 SDC report[2] suggested that approximately 19 man-months were required for the delivery of 1000 machine language instructions. The data were derived from 26 projects, and included program design, coding, and testing time. The incremental time per 1000 additional lines of code was 5 man-months. Corbato's data gathered from the Multics project[3] indicate that productivity can be vastly increased through the use of a higher level language, but software still remains an expensive product.

Much of the high software cost is the result of duplication and of conversion costs. Williams[4] has reported on a conversion project undertaken in 1964 by the Lockheed Missiles and Space Company. The 220 FORTRAN programs which were converted, from an IBM 7094 to a UNIVAC 1108, required five months for the job, at a cost of approximately $241,000. To alleviate this problem we need to ascertain the degree to which higher level language translators conform to published standards, and we could certainly use more in the way of conversion aids, particularly data conversion.

Finally, we have found that the entire competitive selection time can be disturbingly long—nine to 23 months in our experience. We say "disturbingly" because a long selection process is expensive for both buyer and vendor. We are interested in software which could perhaps be used in shortening the time span.

THE PROJECTS

The goals

The user of higher level languages in software development will reduce the cost of such development, principally by increasing programmer productivity. Two languages, FORTRAN and COBOL, have been standardized so as to increase their usefulness. Standardization efforts are also under way for BASIC and PL/1. If standardization is indeed to bear some advantages, commercial compliers must adhere, in their translation, to the published standard. The adherence to a standard must include language semantics (where unambiguous) as well as language syntax. Effective implementation of a standard requires a means of measuring the degree to which compilers conform to the standard. Thus, the development, use, and maintenance of a validation system for COBOL compilers has been an important effort on the part of the Software Development Division.

Portability is a measure of the ease of moving a computer program from one environment to another. Many factors affect a program's portability: the computer system, the language used, program design, and the application. At this time, we are specifically interested in COBOL program portability. A COBOL program would be completely portable if all non-standard functions (e.g., extensions to the language) could be reduced to standard functions, all implementor names could be resolved, data representation were standardized across computer systems, and no "implementor defined" language elements were used. Practically, this means that a completely portable COBOL program is a figment of the imagination! We can, however, greatly improve a program's portability by developing software which addresses itself to the above problems.

It is also important that we not sacrifice too much efficiency for the sake of portability. A recent study by International Computer Systems, Inc.[5] indicates that COBOL programs are generally easier to convert (to other COBOL dialects) than programs in FORTRAN or assembly languages. Unfortunately, the same study also indicates that the relative operating costs of converted COBOL programs are much higher than those for other languages. Our aim is to achieve significant portability at a modest cost in efficiency. Because such an aim is quite relevant to benchmark programs, we refer to the conversion system we are developing as the Benchmark Preparation System (BPS).

A significant factor contributing to delays in computer systems acquisition has been the preparation and processing time of user benchmarks. Some way of measuring minimal system throughput capability is required. For selection purposes, benchmarks are the accepted measurement tool in the Department of the Navy. The major problems with natural benchmarks (i.e., existing application programs) have been the following:

(a) Each time an agency selects a system a new set of benchmarks is prepared. This is wasteful.
(b) The benchmarks are often not debugged, and usually biased toward a given architecture.

The latter problem will be partially alleviated by the BPS. In order to reduce production duplication and costs we are developing a "reference benchmark program library." This is a set of task-oriented synthetic programs which can be used individually or in a mix, in conjunction with or instead of natural benchmarks.

Systems to fulfill the goals

The COBOL Compiler Validation System consists of audit routines, their related data, and an executive routine (VP-routine) which prepares the audit routines for compilation.[6] Each audit routine is a COBOL program which includes many tests, and supporting procedures indicating the results of the tests. The audit routines collectively contain the features of Standard COBOL (except for the Report Writer module). The executive routine automates the creation of a file containing the audit routines with implementor names inserted in the source code, and the operating system control cards required for compiling and executing each routine. The testing of a compiler in a particular hardware/operating system environment is accomplished by compiling and executing each audit routine. The output report produced by each routine indicates whether the compiler passed or failed (individually) the tests in the routine. If the compiler rejects some language element by terminating compilation (giving fatal diagnostic messages) or terminating execution abnormally, then the test containing the code the compiler was unable to process is deleted, and the audit routine compiled again. A test is deleted by inserting NOTE at the beginning of the test paragraph, thereby changing the source code in the test paragraph to comment statements. The output reports of the audit routines constitute the raw data from which the members of the Federal COBOL Compiler Testing Service (an activity of the Software Development Division) produce a Validation Summary Report, which provides a consolidated summary of the results obtained from the validation of a compiler.

The results of running the COBOL Compiler Validation System do not suggest the degree to which the compiler is usable (i.e., capable of data processing applications) but the degree to which individual language elements are usable. This will give an indication of conversions which will be necessary in order to utilize a source program from another system supporting the same language specifications/standard. Thus, the Validation System tests a COBOL Compiler's adherence to the standard language syntax, and, where unambiguous, language semantics. The latter of course is a more difficult area because of the lack of appropriate mechanisms for precise semantic specifications. The Validation System does not evaluate the implementation of a compiler (i.e., is it a text-in-core or compiler-in-core, etc.) nor its quantitative performance characteristics.

Additionally, the summary of a validation includes an indication of unspecified language semantics (i.e., where latitude is given for vendor implementation), and ambiguous language semantics. Finally, tables summarizing the running time and memory utilization of the audit routines, and a characterization of compiler hard copy output and diagnostics are included in Validation Summary Reports.

The benchmark preparation system performs conversion in the major areas affecting portability of application COBOL programs; nonstandard COBOL functions, implementor names, and data representation. A COBOL source program translator (NAVTRAN-C) takes native machine COBOL programs and converts them to machine independent COBOL (ANSI X3.23-1968 language specifications). Those functions in the native machine COBOL which are extensions to the ANSI language specifications (and therefore cannot be converted) are flagged by the translator. Implementor names in the benchmark programs are replaced with unique names in the machine independent source programs. These names are recognized and replaced by the VP-Routine when the programs are implemented on the target machine.

Input data files associated with the benchmark programs are translated by a series of COBOL programs. These data translation programs make use of data conversion subroutines inherent in the respective COBOL Compilers (native or target machine) in translating the machine dependent data to machine independent format and vice versa. Machine dependent data characteristics may include arithmetic sign, word boundary alignment, and certain internal representations. The COBOL data translation programs are created from the benchmark program file descriptions. The creation is performed by program generation. File descriptions in the data translation programs are those for the native machine file, machine independent file, and ANSI/target file. The native machine file description is used to read the native machine data files and build machine independent data files. All data in these will be in display or character mode with the signs of numeric data stored separately. Essentially, machine, dependent data are translated to a string of characters which may then be subject to straight character code translations for the appropriate machine.

Upon transfer of the data files to the target machine, the reverse operation occurs. The machine independent data are read according to the file descriptions, and written using the ANSI/target file descriptions. The data translation programs also provide the capability of validating the data files, e.g., numerically described fields which do not contain numeric data are identified. This can be done by a separate execution or in conjunction with creating the independent or target machine data files. The benchmark package (programs and data files) which is distributed is itself in machine independent form. Programs are in a source program library (Population File). The Population File contains the benchmark source programs, data translation programs and the VP-Routine. Prior to benchmark processing the VP-Routine selects the machine independent COBOL programs from the population, inserts the necessary COBOL implementor names and creates a job stream file for input into the computer system. The VP-Routine also provides the updating capability for the Population File. A summary of all changes made to the Population File and the job control language generated for the run stream file is part of the output created by the VP-Routine. This summary is used to determine the changes a vendor has to make in implementing the benchmark on his system.

The Reference Benchmark Programs Library has been used in performing an experiment to determine the suitability of synthetic programs in alleviating the problems created by natural benchmarks. Five processing tasks were selected as representing, in varying combinations, a large number of application tasks. These were sequential file processing, indexed sequential file processing, relative I/O processing, sorting, and computation. COBOL programs were written to perform each of these tasks, with each program controlled by a set of compile-time and execution-time parameters. The ability to vary automatically certain parameters at compile-time provides us with the flexibility to develop a fairly rich mix from just a few basic programs.

We have found, through our testing with these programs, that a small number of simple, task-oriented, synthetic modules can be combined into a versatile job mix. A relatively small number of parameters is sufficient to enable a single program to reflect the characteristics of a broad class of applications. Also, individual modules have proven useful in exercising isolated computer system features, such as I/O handling. Finally, if one accepts a "modest" workload characterization, aimed more at reflecting extremities and crucial areas rather than comprehensiveness, it is possible and reasonable to construct a benchmark from a set of synthetic modules.

PROJECT CONTROLS

Why

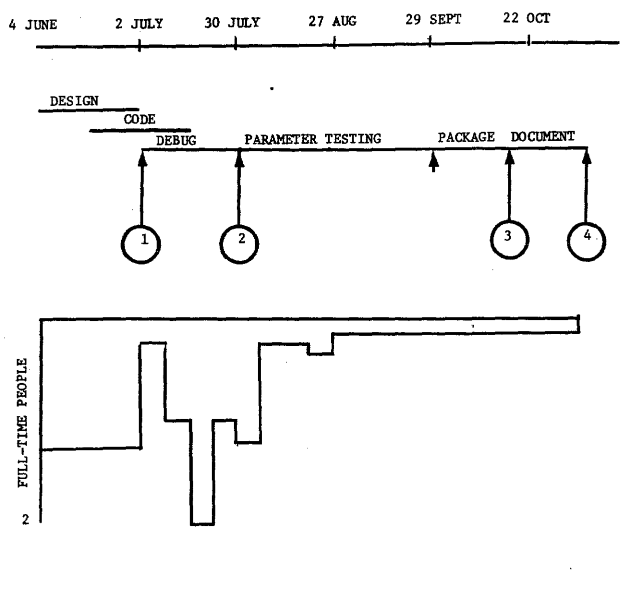
Boehm[1] has suggested that the phrase "software engineering" is a contradiction in terms because we have no data base to be used in measuring, in some way, what we produce and how well we produce it. Yet, his own studies indicate that we *do* have some data to work with. In order to obtain more, those of us whose business it is to develop software must keep records of our efforts, and thereby control them. This does not present an undue hardship in our case since the Department of the Navy strongly encourages that we be accountable for what we do, how we do it, and what it is worth.

How

Because we are a small organization, our controls are modest but, we think, effective.

Much has been made of structured and modular programming.[7] These concepts are gaining acceptance in Government and private industry. While we take no issue with their merits, we would suggest caution in their applicability. Modularity will often reduce some of a system's complexity, but may introduce additional complexity, particularly in the inter-module connections. The nature of the COBOL Validation System dictates that it be highly modular, but we have found that much of its complexity is due to its modularity. We have also found that GOTO-less programming can be awkward and, especially in COBOL, costly. We realize that deviations from the concepts are "allowed," but then we are back to what have for years been recognized as simply good programming practices.

We *do* follow modular programming concepts as design aids. This seems to have become a very common practice. A recent Hoskyns survey[8] for the British Government showed that 98 percent of modular programming practitioners did so in the design stage. A major benefit of this practice has been a lowering of maintenance costs.



Milestones: 1. Submit Narrative
2. Submit Parameter Specifications
3. Complete Population File
4. Complete Instruction Manual

Figure 1—Project history chart for synthetic benchmarks

We've considered the "lead programmer"[9] idea and discarded it as inapplicable to our environment. We are blessed with a surplus of "lead" programmers, and our projects, while sometimes large, as in the case of the Validation System, are not massive.

We keep records of our work. An "initial project form" is used to identify the project requestor, the purpose and nature of the project, time requirements, resources required, and expected payoff. We generally cannot afford the luxury of continuing if resources required exceed payoff, *in dollars*. We then prepare a work plan. This includes a schedule, checkpoints, milestones, and manpower requirement distribution. Milestones are distinguished from checkpoints in that the former require a concrete action or document to be taken or produced, while the latter may simply consist of an indication that "parameter testing is complete". Figure 1 shows the work plan for our synthetic program library project. It is important that we indicate the distribution of manpower over the project lifespan, since this enables us to coordinate manpower requirements for several projects. We review the workplans whenever we feel it is necessary (but at least at the checkpoints and milestones). If we fall behind we revise the workplan. Thus far, we've successfully resisted the temptation to add manpower or adopt unreasonable catch-up schedules when we fall behind. The necessity for such resistance has been well documented by Brooks[10] and others.

We maintained a log of compiler errors (computation, sequence control, input-output, etc.) but we abandoned this because we did not find it overly useful. Over a sample period of five months we produced approximately 10,000 lines of COBOL code; and 42 compile-time errors. Approximately half of these were "clerical" errors (bad keypunching, sloppy printing, etc.). Recognizing the smallness of the sample, we would still make the generalization that any overly extensive effort in beefing up a compiler's syntax diagnostics capability may be a waste of time.

A test log is kept for all projects. The log indicates which program or module is being tested, aims of the test, whether these were achieved, and resources used. The same five month sample showed that we achieved our aims in just under 60 percent of the tests, and that new problems were discovered in some 30 percent of the tests. Also, the average test run used less than three memory minutes of UNIVAC-1108 time. All our work is done in a remote job entry (batch) mode. Yet, the above figures seem to imply that our testing habits are more consistent with what would be expected in an interactive program development environment. Sackman[11] and others have suggested that on-line programming improves efficiency. It appears that, additionally, experienced programmers tend to behave as if they were in an on-line environment, even if they are not.

Boehm's[1] statistics indicate that 45-50 percent of software efforts are devoted to checkout and testing, and that only about 20 percent of the time is spent in coding. The data base for these figures was derived from large systems projects, such as the OS/360 development. Ours are much more modest projects, and our results are both different and more variable. About 50 percent of our time in the synthetic library project was devoted to coding, and less than 25 percent of the time was spent on integration and testing. The Benchmark Preparation System figures are quite different. Coding has taken up less than 25 percent of our time, with integration and testing using up some 60 percent of the time. The resulting low figure for analysis and design (15 percent) is due to the simple fact that much is already known about the portability problem areas in COBOL.

Packaging and distribution of all our software products follow fairly simple guidelines. The programs, in machine independent form (all implementor names are parameterized, as are machine dependent features such as precision and size of numerical fields), are placed on a standard magnetic tape reel, together with a copy of the VP-Routine. The latter is used for parameter substitution (to a form acceptable to any specific system), "library management", and job control statements generation. Accompanying the tape is a user guide, brief narrative description of the system, and, where applicable experimental results. The programs are self-documented, so that we can avoid excessive external documentation. While we recognize the importance of *adequate* documentation, we have found that excessive documentation, such as detailed flow charts can be a hindrance to proper documentation. Distribution is through the National Technical Information Service.

A few words of caution about these and other published statistics and practices. First of all, they reflect a very specific environment. We have a small (eight people) staff with very homogeneous backgrounds. Our systems are modest in size, and "utility" oriented. All our work *must* be portable, since we are currently using UNIVAC-1108, IBM 360/65, and HIS 6050 systems for product development. Furthermore, our COBOL compiler validation responsibilities have recently required us to use our software on a Burroughs 6700, HIS 437, and IBM 370/155 system. Thus, portability is truly a necessity for us.

Secondly, even for a similar environment, the statistics should be viewed as "guidelines". They are simply products of our experience which we hope to learn from but do not expect to be bound by.

THE PAYOFF

What has it all cost us?

Total cost for the synthetic programs library, including machine time, clerical support, and salaries was under $6,000. This benchmark preparation/conversion package has cost us about $8,000. The COBOL Compiler Validation System was originated in 1969 by the U.S. Navy Programming Languages Group under the direction of Capt. Grace M. Hopper, USNR. A reasonable estimate of its initial cost is not possible, but we do have an expected cost for the audit routines we are preparing in anticipation of the revised COBOL standard. Our schedule calls for completion of the project by November, 1974. Total calendar time for the project will be 15 months and we anticipate to expend 36 man-months on the effort. Total cost for the new Validation System should be in the neighborhood of $75,000. The new system will be approximately twice the size of the current one, which is comprised of about 130 programs, or 100,000 lines of COBOL code. The implication here is that we expect our productivity to be about 33,000 lines of COBOL code per man-year; a remarkably high figure (Corbato[3] has reported a number in the neighborhood of 1200 PL/1 lines of code per man-year on the Multics project). This is due almost entirely to the fact that we are "borrowing" most of the design work from the present Validation System. We know the modules we will require since the standard is defined for us. The VP-Routine is already available. Many of the audit routines will be extensions of current ones. Thus, our time will be spent primarily in identifying tests, coding, and testing.

What are the benefits?

We expect the returns on our investment to be substantial. The best COBOL compiler we have tested to date ("best" in its conformance to the standard) has had some 30 areas of non-conformance. This not only impacts portability, but can have serious side effects. Many data base management systems are COBOL-based. Errors in a compiler can easily result in "dirty" data getting into the data base. We have, for example, identified some four different treatments of arithmetic statements, each producing different results! The validation of a compiler tells us where the danger areas lie. Furthermore, vendors are required to correct discrepancies once these have been identified. Thus, our validation of COBOL compilers enables us to reap the benefits of standardization. Without such a measurement tool standardization is a fruitless endeavor.

The high costs of processing benchmarks has already been mentioned. We know of a recent selection where the total award was for approximately $5 million. Included in that figure were some $500,000 which the vendor spent in processing the benchmark. Both the benchmark preparation system and our synthetic programs library would pay for itself if even a small portion of these potential savings in vendor expenditures are passed back to the Navy.

FUTURE EFFORTS

The benefits derived from validating COBOL compilers would also accrue in the validation of compilers for other languages. FORTRAN, BASIC, and, later, PL/1 are natural candidates.

Compiler efficiency, in terms of object code execution speed and storage utilization, has a significant impact on an installation's throughput. This in turn affects the timing of selections and therefore of expenditures. We believe more efficient compilers mean fewer dollars spent, or more work done for the same dollar. Thus, we are planning a set of test routines to determine the *relative* worth of a given compiler. That is, we want to measure how much room there is for improvement in execution speed and storage required. This project is in the design phase and will restrict itself, initially, to FORTRAN compilers, principally because it is easier to measure efficiency of FORTRAN compilers than those for most other languages. Knuth's work[12] suggests that our efforts may prove fruitful.

Finally, we believe that serious thought must be given to validating generalized data base management systems. Specifically, we are interested in finding ways of ascertaining that the data base one builds with these systems does indeed contain what we wish it to, that in retrieving data we get all that is proper, and *only* what is proper, and that use of such systems does not impact the integrity of the data base. We also plan to develop simple analytical models to be used in evaluating different types of data organizations. The possible ongoing contamination of these data bases by inconsistent object code has already been commented on.

CONCLUSION

Software to be used in improving or measuring the quality of other software is neither difficult nor expensive to produce. Our efforts are concentrated in the system selection area. We believe, however, that the benefits to be derived from such efforts have a broader scope, and are substantial enough to warrant persual by any data processing organization.

REFERENCES

1. Boehm, B. W., "Software and its Impact: A Quantitative Assessment," *Datamation*, May, 1973.
2. System Development Corporation, *Estimation of Computer Programming Costs*, SP-1747, September, 1964.

3. Corbato, F. J., "PL/1 as a Tool for System Programming," *Datamation*, May, 1969.
4. Williams, D. A., "Conversion Case Study and Experiences," American Management Association, Administrative Services Briefing Session #6379-02, *"A Hard Look at Software"*.
5. International Computer Systems, Inc., *Programming for Transferability*, AD-750 897, National Technical Information Service, 1972.
6. Baird, G. N., "The DOD COBOL Compiler Validation System," *Proceedings FJCC*, 1972.
7. Liskov, B. H. and E. Toster, *The Proof of Correctness Approach to Reliable Systems*, The MITRE Corporation, ESD-TR-71-222, Bedford, Massachusetts, 1971.
8. Rhodes, John, "Tackle Software with Modular Programming," *Computer Decisions*, October, 1973.
9. Baker, F. T., "Chief Programmer Team," *IBM Systems Journal*, Volume II, No. 1, 1972.
10. Brooks, F. P. Jr., "Why is the Software Late?," *Data Management* August, 1971.
11. Sackman, A., *Man-Computer Problem Solving*, Auerback Publishers, Inc., 1970.
12. Knuth, D. E., "An Empirical Study of FORTRAN Programs," *Software-Practice and Experience*, Volume 1, John Wiley and Sons, 1971.

Experiences in COBOL compiler validation

by GEORGE N. BAIRD and MARGARET M. COOK

Department of the Navy
Washington, D.C.

INTRODUCTION

The Federal COBOL Compiler Testing Service (FCCTS) is an activity of the Software Development Division of the Department of the Navy, Automatic Data Processing Equipment Selection Office (ADPESO). Since July 1, 1972, all COBOL compilers brought into the Federal Government have to be identified as implementing one of the levels of the Federal COBOL Standard. The National Bureau of Standards, which has the responsibility for the development and maintenance of Federal ADP Standards, has delegated to the Department of Defense, and thereby to ADPESO, the responsibility for the operation of a Government-wide COBOL Compiler Testing Service. This responsibility is discharged by the FCCTS through the implementation and maintenance of the COBOL Compiler Validation System,[1] a comprehensive set of routines used to test COBOL compilers for compliance with the Federal COBOL Standard as prescribed in Federal Information Processing Standards Publication 21 (FIPS PUB 21),[2] published by the National Bureau of Standards.

This paper addresses several questions that arise in compiler validation. Why validate compilers for conformance to a standard? How is the validation performed? What experiences have been gained, and what conclusions can be derived from them? The questions will be discussed in turn.

WHY VALIDATE COMPILERS?

The purpose of validating a compiler is to ensure that syntactically correct programs compile and execute without abnormal termination, and that the semantics of the language being translated are correctly interpreted. Also, (where appropriate to the language), validation should point out the impact of implementor defined specifications which are allowed by the standard.

There are three phases in the computer systems acquisition cycle during which a validation is important. Prior to selection, a validation of the compilers for the various systems being proposed may constitute a part of the systems evaluation. After a computer system has been selected, a validation of the present compiler and a validation of the compiler to be procured disclose the areas of nonconformance in both compilers. The effort required in converting existing programs to the new system can then be realistically estimated prior to the changeover. After the delivery of a new computer system, but prior to acceptance, a compiler validation will reveal areas where a compiler does not meet the terms of the contract.

Our experience with the Validation System has shown that continuing benefits accrue from being aware of a compiler's status vis-a-vis its language standard. Compiler validation for computer systems which have already been acquired and are in present use can serve to point out which language elements do not operate correctly and therefore should not be used. A validation is particularly useful when a new version of a compiler or operating system is released, since it will immediately reveal errors in the revised software.

If a user has access to several different computer systems and is doing program development on all of these, he must know what language elements conform to the standard on each of the systems. Validation of the compilers on each system shows which language elements perform correctly. By writing programs using only these elements, a user ensures program portability.

SCOPE OF VALIDATION

Errors in compiling a program may arise from a single statement or a particular sequence of statements. Since validation verifies that individual language elements are processed correctly, errors in combining language elements may exist even though each of the separate elements are processed correctly.

A validation is not concerned with the efficiency of the object code generated, but only tests if the code is produced correctly. A validation system does not test implementor extensions to the language. If the implementor extensions cause problems in the standard language elements, a validation will identify these errors, but any errors in the use of the language extensions themselves will not be discovered during validation.

Finally, while a validation identifies problem areas in the use of standard language elements with a given compiler, it cannot indicate the ramifications of the compiler errors discovered.

FCCTS COBOL COMPILER VALIDATION

The validation of COBOL compilers by the Federal COBOL Compiler Testing Service is performed using the COBOL Compiler Validation System (CCVS), which was developed by the Department of the Navy Programming Languages Section under the direction of Capt. Grace M. Hopper, USNR. The CCVS consists of audit routines, their related data and an executive routine. Each audit routine is a COBOL program, and includes many tests of individual language elements. Supporting procedures indicating the results of the tests are included in each routine. The audit routines of the CCVS collectively contain the features of Federal Standard COBOL.

There are certain adjustments which must be made before the audit routines can be compiled upon a given computer system. First, names allowed by the COBOL standard to be implementor defined must be inserted into the audit routines before they can be compiled. Second, system control cards are required in order to compile and execute a COBOL program. File specification and allocation are also regulated by system control cards, and additional control cards are usually required by programs using the COBOL SORT verb. Third, the given system configuration may not include a hardware device or capability required by some of the test procedures in the CCVS, for example a system which does not support multiple unit assignment for mass storage devices. All references to multiple units must be deleted for the proper compilation of the audit routines on that system.

EXECUTING THE AUDIT ROUTINES

Input parameters to the CCVS executive routine[3] specify the implementor names, hardware dependent language elements to be deleted, and the operating system control cards required to compile the audit routines on a given computer system. The executive routine creates a file containing the audit routines with implementor names inserted in the proper place in the source code, and the operating system control cards required for compiling and executing each routine.

The audit routines in the CCVS consist of source code which is syntactically correct; the routines do not contain any tests which deliberately introduce incorrect syntax. Thus, each audit routine is expected to compile without errors. (This is frequently not the case. We have encountered "Standard" compilers where syntactically correct source code causes fatal diagnostic messages, compiler aborts, and even compiler loops.)

When an audit routine does not compile, or complete execution normally, the source code containing the language elements which the compiler could not handle is modified or deleted. The tests in the PROCEDURE DIVISION of the audit routines are coded so that a test is deleted by inserting NOTE at the beginning of the paragraph containing the test. This results in the entire paragraph being treated as a comment. The source paragraph following the deleted paragraph contains procedures which indicate the test has been deleted.

The CCVS executive routine contains an editing capability which permits addition, deletion, or replacement of source lines in the audit routines. After an audit routine has been modified so that it consists of only the language elements that the compiler accepts, the routines are again compiled and executed.

All of the supporting procedures for verifying whether a test passes or fails is contained in each routine. An output report is produced indicating the actual results of each test, and, when a test fails, the expected result.

ADDITIONAL INFORMATION FROM COMPILER VALIDATION

A compiler validation identifies many characteristics which can be used in comparing compilers. Due to compiler errors, valid syntax in the audit routines may be rejected or the resultant object program may abort during execution. The running of the CCVS on a system supplies information concerning the effectiveness of diagnostic messages. The effort required in locating the source code which caused execution to terminate abnormally can also be assessed.

The procedures used in validating a compiler give information on a system's ability to execute a program after fatal compilation errors have been diagnosed, or the effectiveness of a system supplied option for skipping execution if fatal errors are encountered.

Some minor programming aids may also be discovered during a validation. The audit routines can be compiled using options for cross reference listings of data-names and procedure-names. Some compilers flag blank cards. In some cases, we have uncovered hindrances. The suppression of the printing of the contents of columns 73 through 80 on a source listing is, for example, a hindrance to anyone running the CCVS, since the CCVS executive routine uses columns 73-80 to indicate whether a source line has been replaced or added.

THE VALIDATION SUMMARY REPORT

The output of a validation is a set of listings from the executive routine documenting the steps taken in preparing programs/jobs for execution; the compilation of each program; and the execution report of each program. These listings constitute the raw data from which a Validation Summary Report (VSR) is produced. (Any attempt to use the raw results for evaluating a compiler would be painful indeed due to the volume of paper involved.) The VSR provides the following information:

(a) The status of the compiler in relation to each of the four Federal levels of COBOL as defined in FIPS PUB 21.
(b) A list of language elements whose implementation

is not consistent with the language specification (American National Standard COBOL X3.23-1968).[4]

(c) A list of language elements which are not implemented due to a lack of hardware necessary to support those elements (e.g., read reversed, hardware switches, etc.). The language specification does not require the implementation of certain elements if they are dependent on specific hardware devices, and the system supporting the COBOL compiler itself does not support that device.

(d) Information-only items. These are necessitated due to the existence of imprecise language specifications in the Standard.

(e) Compiler characteristics noticed by the validation team. This includes the usefulness of diagnostic messages issued, format of the source program listing, and timings and memory requirements for the compiler and individual audit routines.

COBOL COMPILER PROBLEMS DISCOVERED DURING VALIDATIONS

The FCCTS has, since its establishment, validated compilers supplied by many different vendors. Many problem areas have been uncovered during these validations. Some of the problems are common to many compilers; others occurred only in particular ones.

In this section we present some of the common problem areas we have discovered in "Standard" COBOL compilers. The problems are grouped by the COBOL functional processing module in which they are found.

Nucleus

In a NOTE character string, any combination of characters from the computer's character set is treated as a comment; the string appears on the source listing, but is not compiled. From the NOTE tests included in the CCVS we have found that most COBOL compilers check the syntax of a NOTE statement. As an example, one of the tests is a NOTE sentence containing a single " (QUOTE) character. Most compilers generate a message indicating an illegal alphanumeric literal format was used since an alphanumeric literal is a character string enclosed in quotes. If the NOTE statement is not the first sentence of a paragraph, the NOTE comment ends with the first period followed by a space. On many systems though, the first period, with or without a trailing space, terminates the NOTE comment and attempts are made to compile the rest of the comment.

There are tests of the ADD and SUBTRACT CORRESPONDING statements with matching items requiring five levels of qualification. Most compilers give diagnostic error messages for these tests, but in some cases the compilation of the program was terminated.

A test of the floating insertion editing capability included in a Nucleus module audit routine moves 000123.45 to an elementary item with a PICTURE clause $$$B999.99. The language specification states that any of the simple insertion characters (comma, blank and zero) immediately to the right of floating insertion characters are part of the floating character-string. Thus, the contents of the edited item after the test should be $123.45. The result usually obtained for this test is $b̸123.45.

Sequential and random access

The size of the data records for a file may be specified by the RECORD CONTAINS integer-1 TO integer-2 CHARACTERS clause. The size of each record in the file is defined by the individual record descriptions and the RECORD CONTAINS clause is optional. If the clause is used, there should not be any restrictions in the File Description. We have found compilers which require the number of characters in each record to be a special item at the beginning of a record description when this clause is used. This is a nonstandard restriction.

One audit routine for both sequential and random access file processing includes a CLOSE file WITH LOCK statement. Attempts are then made to OPEN and READ the locked file during the current run unit. A file that has been closed with lock cannot be opened again during execution of the object program. Most of the systems tested abort the program execution with an error message stating that an attempt to access a locked file has been made. Some of the systems return data and continue executing as if the CLOSE WITH LOCK had not been encountered. In one case, the program went into an execution loop when an attempt to read a locked file was made.

Table handling

In the COBOL Table Handling Module, the OCCURS DEPENDING ON option specifies a table whose number of occurrences varies during execution. The number of items in the table during execution depends on the value of the data-name in the DEPENDING ON clause. If a SEARCH statement is encountered for a variable table, the last entry in the table is defined by the contents of the data-name. Some compilers accept the syntax for the table definition, but the table is always considered to be its maximum size in a SEARCH statement.

Segmentation

An independent program segment is expected to be in its initial state each time the segment is made available to the program. There are compilers which do not restore the initial state of independent program segments.

Library

The COPY statement with the REPLACING option allows the user to copy library text, replacing each occurrence of a word in the text with a new word. A variety of problems have been found when exercising this option in the audit routines. One compiler correctly handled a COPY REPLACING statement, but in a subsequent COPY without the REPLACING option of the same library text, words were replaced which should not have been. This caused undefined data-names during the program compilation.

Many compilers place restrictions on the REPLACING option which are not in the Standard. Some of these restrictions are that a qualified name could not be replaced; a data-name could not be replaced by a subscripted data-name; or a data-name could not be replaced in an 01 level entry in the Data Division.

Sort

A frequent restriction encountered in the SORT module is a limitation in the specification of the minimum number of characters in a sort record description. Usually the compilation of the COBOL source program will not cause any diagnostic messages, but when the sort program is executed, a message indicating incorrect record length is produced.

Problems have arisen when one of the sort keys is defined as a signed numeric field. For a sort on ascending keys, negative values should appear before positive values. In some cases, signed numeric items have not been sorted in the correct order because comparison was based on the actual binary structure of the data, and not the algebraic value associated with the data.

Implementation variations

A file may contain multiple record descriptions, with each record having a different length. It is important for a user to know how much external storage media must be allocated for records of a given file. In order to compute this a user must know if a given implementation always writes records of the maximum length, or if variable length records are written. This would be especially significant if most of the records in the file were much shorter than the remainder. Though not required by the standard, a great deal of external storage space is wasted if fixed length records are written.

There are procedures in both sequential and random access audit routines which create a file containing variable length records, and in the subsequent reading of the file test if the system creates all records as the maximum fixed length.

As a result of our validations, we have changed some tests to information-only tests because of language ambiguities. For example, there is no explicit statement in the Standard as to what the result should be when moving a signed numeric field to an alphanumeric field. There are statements suggesting that any appropriate conversion takes place during an elementary move, but there could be doubt as to whether the elimination of a sign is included. In order to determine the result for a particular compiler, the alphanumeric item is tested for a numeric value after moving a $+1$ and a -1 to the alphanumeric item.

The use of optional words is only for readability and they are not supposed to effect the meaning of the statements. However, we have found one case where the presence of the optional word 'IS' changes the meaning of a statement. One of the audit routine tests is

IF A = B AND IS NOT GREATER C OR D.

The presence of the word 'IS' leaves no doubt that NOT is part of the relational operator NOT GREATER. As a result, the expansion gives

IF A = B AND A IS NOT GREATER C OR *A IS NOT GREATER D.*

But for the combined abbreviated relational condition:

IF A = B AND NOT GREATER C OR D,

the language specification states explicitly that the word NOT is part of the logical connector AND NOT, and is not part of the relational operator NOT GREATER. The effect of the rule causes the statement to be expanded so that the NOT is associated with the operand called C, and not associated with the operand called D:

IF A = B AND NOT A GREATER C OR *A GREATER D.*

The presence of the word 'IS' in the statement can indeed violate the law of least astonishment.

The discovery of problems in tests such as those described above has resulted in recommendations for language clarification to the American National Standards Institute Technical Committee X3J4, which has the responsibility for the maintenance of X3.23-1968 COBOL.

REASONS FOR FAILURES IN EXECUTING THE AUDIT ROUTINES

There are several reasons why a compiler may be implemented in such a way that there are discrepancies between the language specifications and the results given by the compiler. The most common reason for discrepancies is simply due to logic errors in the compiler. This can be attributed to lack of adequate controls in producing compilers.

A second reason for differences in the implementation is the misinterpretation of the language specification on the part of the implementor. One case in point (which has been noticed in several compilers) is the use of the word THRU in the current standard (X3.23-1968). In many specifications throughout the document, the word THRU appears as follows:

PERFORM procedure-name-1 THRU procedure-name-2 FILE-LIMIT literal-1 THRU literal-2, etc.

There is nothing in the above syntax which indicates that the words THRU and THROUGH are interchangeable.

The only reference that establishes this little known fact is the reserved word list where they are shown to be equivalent. This probably accounts for the fact that we have identified several compilers which do not allow usage of THROUGH.

Another problem is the ambiguity that is inherent in a language as complex as COBOL. These are areas in the current language specification that, at best, are ill-defined, and the implementor must make a unilateral decision as to the direction the implementation will take. A good example would be the default action the compiler takes for a WRITE statement to the printer, when the ADVANCING phrase is not specified. Based on whether the default assumes WRITE BEFORE ADVANCING or WRITE AFTER ADVANCING inappropriate spacing or overprinting of lines can occur. This was recognized as a shortcoming of the language, and subsequently corrected so that a default is now specified.

RESOURCES REQUIRED

An accounting summary is prepared for each validation performed. This summary includes professional personnel time for required modifications to the CCVS (to accommodate any compiler peculiarities), site visit for acquisition of raw data, evaluation of raw data, and preparation of the VSR; and processor time required for executing the audit routines. An average validation has thus far required 73 hours of professional time and 29 hours of clerical time. The processor time will naturally vary with the computer used. Execution of all the audit routines takes approximately 20 minutes of UNIVAC 1108 processor time. We estimate the average cost of a validation to be approximately $1,000, although these figures are subject to wide variations.

CONCLUSIONS

The use of the CCVS in validating COBOL compilers attempts to answer three major questions:

- Will the compiler accept the syntax as defined in the COBOL language specification?
- Will the compiler generate the appropriate object code to satisfy the semantic requirements of the language specifications?
- What unilaterial actions does the compiler take when the language specification leaves the result up to the implementor?

The results of over a year's work in validating a variety of compilers indicate that there may not be a compiler which completely conforms to the COBOL standard. In a few cases we were tempted to question whether the compiler was in fact compiling COBOL, or some other language similar to COBOL!

The reasons for the amount of non-conformance or deviation from the language specification can be blamed partly on the 1968 COBOL Standard. Most of these problem areas have been resolved during the development of the revision to X3.23-1968. As a result, we feel that we can expect to see better compilers since the language specifications are tighter and better defined; the idea of providing standard compilers is being encouraged in the marketplace by the users; and, most importantly, we have a measurement tool which can be used to determine the degree to which a compiler conforms to the Standard.

We recognize that the Validation System is necessarily incomplete. But we also are convinced of the importance of having some capability for measuring the quality of software. What we have learned during the period we have been validating compilers confirms the importance of software engineering, and thereby the importance of any measurement tool which results in software quality improvement.

REFERENCES

1. Baird, G. N., "The DOD Compiler Validation System," *Proc. 1972 FJCC*, AFIPS Press, Volume 41, Pages 819-827.
2. *Federal Information Processing Standards Publication 21*, U.S. Government Printing Office, Washington, D.C., March 1972.
3. *Navy COBOL Compiler Validation System User's Guide*, Information Systems Division, Department of the Navy, January 1973.
4. *American National Standard COBOL X3.23-1968*, American National Standards Institute Incorporated, New York 1968.

System for efficient program portability

by GEORGE N. BAIRD and L. ARNOLD JOHNSON

Department of the Navy
Washington, D.C.

BACKGROUND

The acquisition of major Automatic Data Processing Equipment (ADPE) systems in the Department of the Navy is accomplished by either single source acquisition or competitive selection.[1] In the case of the single source acquisition, only one vendor is considered as having the system capable of satisfying the needs of the procuring organization. The criteria which must be satisfied for a single source acquisition include the need for unique hardware, excessive conversion/reprogramming cost and/or time, etc. The competitive selection, on the other hand, is open to any vendor who feels he can provide a system meeting the requirements specified in the solicitation document associated with that procurement.

The major steps in the competitive process are, briefly, as follows:

(1) The requestor prepares a solicitation document which explains the requirements of the system being procured.
(2) Benchmark programs are prepared. These will be used to ascertain that vendors participating in the competition can meet minimal performance standards. These programs may be written in various higher level languages. We will restrict our discussions to COBOL programs.
(3) The vendor examines the solicitation document to determine whether he has a system capable of meeting the requirements.
(4) The vendor obtains the benchmark, converts it to the system being considered, and determines if the hardware/software will be price-competitive with the systems most likely to be offered by his competitors.
(5) The vendor must then demonstrate that the execution of the benchmark can be accomplished within the time specified in the solicitation document.
(6) The award is generally made to the vendor who qualifies with respect to timed benchmarks and offers to provide the system which meets the needs of the user at the lowest overall cost to the Government.

The usual amount of time involved in the competitive selection process, as described above, ranges from nine to 23 months. A significant portion of this time is involved in working with the benchmark. The system we are describing here is designed to reduce this time.

USE OF BENCHMARKS IN THE SELECTION PROCESS

The benchmark is a vital part of the competitive selection process. It becomes the tool for minimum measurement to be used against all systems being considered. Therefore it is important that the benchmark be constructed in such a way as to accurately reflect the system requirements being specified. The system requirements are defined in terms of the current workload and the projected future workload. Properly prepared benchmarks will demonstrate that the system being offered contains adequate memory and peripherals, and that the throughput speeds are sufficient to process the projected future workload. Additionally, this exercise demonstrates that the operating system and supporting software are operative. From a functional standpoint, the "ideal benchmark" situation could be described as follows:

(1) The programs would be coded using the language elements defined in the American National Standard COBOL, so that source code conversion is minimal.
(2) The implementation of the benchmark programs by the various vendors could be monitored as in a controlled environment. This would provide useful information as to the impact of converting other programs after the new system is delivered.
(3) The programs have been debugged to the extent that they will give predictable results on both the native and the target machines.
(4) The data files would be in a form readily acceptable to all systems, but would at the same time be consistent with any given system's architecture, so that there is no loss of efficiency, or validity of results.
(5) The checking of benchmark processing results would be as automated as possible.

Unfortunately, this ideal situation seldom exists. This results in excessive time and cost expended on the part of the vendor in processing the benchmark. It is not unusual for a vendor to spend six to nine calendar months just preparing the benchmark programs for processing, or for the cost of processing them to represent 10 percent or more of the eventual bid price. The cost and time related to processing a benchmark causes vendors to be more selective in responding to requests for proposals (RFPs). This could result in the best system not being offered if the vendor feels he may not have a good chance of winning.

PREVIOUS BENCHMARK EFFORTS

Little has been done in the area of making the benchmark, and, as a result, the competitive process, more palatable. In the past, various methods have been used in presenting the benchmark to the vendor. These range from the attitude of "here is the benchmark, do with it what you must in order to make it run on your system, and in the meantime don't bother me" to a recent instance where the benchmark and its related data were provided in what could be described as "machine interchangeable" form (i.e., all data was in DISPLAY form, several "dangerous" language features were not used, etc.). This approach is consistent with recommendations made by Meltzer and Ickes.[2]

The effect of the "don't care" philosophy is that the vendor is permitted to make any changes he desires in implementing the benchmark. This may destroy the representativeness of the benchmark. At the same time, through the use of his most talented programmers and/or analysts, a vendor could optimize the programs for faster execution. It is often the case that the talent the vendor is able to turn loose on the benchmark is superior to that of the average user organization. Therefore, the credibility of the benchmark is somewhat lessened, and the timing results now represent more what the vendor's programming staff can do than what was originally intended.

The philosophy behind the machine interchangeable COBOL calls for no modification being permitted to the source programs except in the Environment Division. Basically, the idea of machine interchangeable COBOL is to eliminate any form of data representation except for standard data format (e.g., character representation only—no binary, packed decimal, floating point, etc.).

The machine interchangeable approach satisfies the majority of the criteria described in presenting the ideal benchmark; however there are two comments worth making:

(1) The decision to stay with the standard data format may force several vendors not to participate; primarily those with systems not capable of character addressing and/or decimal arithmetic.
(2) The resources used in manually converting the benchmark package to machine interchangeable format will include a substantial amount of manpower and computer time.

THE SANITATION EFFORT BY THE U.S. NAVY

The Software Development Division of the Department of the Navy Automatic Data Processing Equipment Selection Office (ADPESO) is looking into methods and techniques for decreasing the amount of time required for competitive selection, and lowering the overall cost of the procurement. Armed with the knowledge of problems associated with benchmark techniques, and in particular the problems associated with natural benchmarks, an effort was established to mechanize the preparation of natural benchmarks.

The vehicle we chose (for ease of implementation, with the necessary documented controls) was the VP-Routine developed by the Navy for automatically resolving implementation names within COBOL programs, and generating the necessary operating system control cards to compile and execute the programs.[3]

Each source program must be purged of nonstandard language elements. This is accomplished by automatic conversion (where possible through simple syntax replacement) and hand tailoring when additional logic may be required to handle semantic differences between two statements. Also, all implementor names must be resolved, or converted to an intermediate form. This results in COBOL source programs that are VP-Routine sensitive in that they can be tailored to a given systems requirements by a single pass of the VP-Routine.

All data files necessary for input to benchmark programs, or output files provided for verification purposes, must be carefully extracted from the native system and transformed into a form readily acceptable to other systems. This must be done with no loss of data integrity.

After the source programs and data have been converted, it is important to insure that the execution of the converted benchmark programs give the same results as the original programs, and that the data has not been adversely affected. This is accomplished by executing the sanitized benchmark using the sanitized data on both the current computer system and on other target systems. During the execution of the benchmark it is useful to be able to determine the degree to which the data is exercising the various procedures of each program. Through the help of a software monitor, information is provided which will help to further determine the adequacy of the benchmark.

The result of the benchmark preparation process would be a viable product that is:

(1) Well documented.
(2) Checked out on more than one system.
(3) Easily implementable through the use of the VP-Routine.

A discussion of the overall system follows in five major sections:

Source Program Preparation
Data Portability
Program Monitor
Packaging and Distribution
Limitations

SOURCE PROGRAM PREPARATION

The source programs that are to make up the benchmark are processed by a COBOL to COBOL translator that performs three major functions.

(1) The Environment Division is rendered VP-Routine sensitive by the replacement of all implementor names with an encoded mnemonic that has meaning to the VP-Routine. These mnemonics are later used in preparing the programs for a given system.
(2) The source code is examined for nonstandard coding, and translated when appropriate. Where translation cannot be accomplished, the translator flags the offending code.
(3) Based on parameter cards, the file descriptions of the files to be converted from the native system are used to create an intermediate work file containing pseudo file/record/field descriptions. This file is later used to create Data Translation programs. This is fully discussed in the section entitled Data Portability.

The translation of one COBOL dialect to another is conceptually simple. The more serious problem involves the moving of data from system to system. A detailed discussion of this problem and a suggested solution follows.

DATA PORTABILITY

Problem

The differences in the internal representation of data among computer systems represent the major limitation of software portability. These differences can be broadly segmented into two categories: Differences in the character code used (i.e., EBCDIC, BCD, FIELDATA, etc.), and differences in the representation of numeric data. Data translators for character code conversion are widely available, or can be easily created. Transferability of the second type requires more than simple code conversions, and, therefore, presents the greater problem. Furthermore, the specific representation used within this general category will, for a given computer, seriously impact the effectiveness with which that computer is used. Thus, the system described here concerns itself only with the conversion of noncharacter data from any "native" computer to any "target" computer, in such a way that the target computer architecture is properly utilized.

Generally, COBOL data portability is impacted by variations in numeric data representation, alignment of data within a defined data unit (e.g., a word), and the position and representation of arithmetic signs. There are several forms of numeric data representation, of which binary and packed decimal are the most common. The packed decimal format is not universal and may therefore have to be converted to a completely different type of data structure. Data in binary representation *are* universal, and although sign conventions and word size do vary, conversion between binary representations is relatively simple. Alignment variations affect the positioning of data within storage units, particularly in word-oriented computers. For proper transformation of data from external storage to internal storage, the COBOL program definition of this data must be consistent with the expected data position and alignment on the external files.

Solution

Generality and *impartiality* are principal design goals of our effort. Our system must perform data translation from *any* given system to any other system. Furthermore, we must not penalize the architecture of the target system. Generality implies that the translation programs must be automatically generated, as opposed to hand-coded. Impartiality means we must go from a machine dependent form to another machine dependent form. Good sense suggests we do this through an intermediate machine independent code.

We use available software as much as possible. This is done by using the code conversion subroutines already present in a system's compiler, together with the data descriptions in the COBOL programs being converted. The data translators which constitute the heart of the system are automatically generated COBOL program segments. These data translators are used to convert native machine dependent data (MDD) to standard data format (SDF), and the latter to target machine dependent data, which we refer to as machine ANSI data (MAD). If character code translation is required, it can be performed on the SDF, since this data is simply a string of characters.

Program creation

Data translation/verification programs (henceforth referred to simply as *data translators*) are created from the file/record descriptions of the COBOL programs being converted. The data translators will contain the following COBOL file descriptions (FD's):

(1) Machine dependent data (MDD) file descriptions, which are those used to process the file on the native machine.
(2) Standard data format (SDF) file descriptions, in which all data items are in DISPLAY mode, unsigned and unsynchronized.
(3) Machine ANSI data (MAD) file descriptions, in which all data items are described in Standard COBOL formats.[4]
(4) Machine ANSI data for the target machine (MADT) file descriptions. This file description is identical to (3) above, and is used for file comparison purposes. This comparison process is more fully described below.

Source code MDD item descriptions which are not Standard COBOL will be defined by the MAD file description in a form which is as close to the native file description as possible (e.g., COMPUTATIONAL-3 will become COMPU-

```
            01 MDD-RECORD.
               02 ALPHANUMERIC-D      PICTURE X(20)    JUSTIFIED RIGHT.
MDD            02 UNSIGNED-D          PICTURE 9(6)     COMPUTATIONAL-3
               02 SIGNED-D            PICTURE S9(6).

            01 SDF-RECORD.
               02 ALPHANUMERIC-X      PICTURE X(20).
SDF            02 UNSIGNED-X          PICTURE 9(6).
               02 SIGNED-S            PICTURE X.
               02 SIGNED-X            PICTURE 9(6).

            01 MAD-RECORD.
               02 ALPHANUMERIC-A      PICTURE X(20)    JUSTIFIED RIGHT.
MAD            02 UNSIGNED-A          PICTURE 9(6)     COMPUTATIONAL
               02 SIGNED-A            PICTURE S9(6).
```

Figure 1—Example of record description used in a data translator program

TATIONAL). Figure 1 illustrates a DATA DIVISION from a data translation program. Procedures for translating from one data form to another and for file comparisons (for data verification purposes) are generated for each elementary field of the record. There are three data translation procedure types, corresponding to alphanumeric data, signed numeric data, and unsigned numeric data. The type of procedure generated is based on the elementary COBOL item description.

Data translation procedures for alphanumeric and unsigned numeric data require no more than a COBOL MOVE statement. Any changes in the data code, data alignment, or storage allocation required in converting from one form to another are performed automatically by the code generated (for the MOVE statement) by the compiler being used.

To take into account the various sign conventions, COBOL procedures are added to check the characteristics of signed numeric data. If translation is from a machine dependent form to a SDF form, the appropriate sign is stored as a separate character in the SDF data description. If we are performing the reverse process, we first generate the positive value of the machine dependent data item (through a MOVE statement), then check the separate sign character in the SDF description, and if minus multiply the machine dependent data item by minus one to give it the correct sign.

Data validation and verification procedures are also generated. Figures 2 and 3 give examples of the various COBOL procedures used for data translation, validation, and verification (the latter two functions are described below).

Once all the procedures and file descriptions have been generated, they are merged with appropriate housekeeping COBOL statements, resulting in data translators which are complete COBOL programs in VP-Routine sensitive format.

Program operation

Since the native source code file descriptions in the data translators may not be acceptable on the target compiler, the VP-Routine is used to provide the capability of selecting the appropriate source coding for use on the desired system (target or native). This is accomplished by parameter cards to the VP-Routine. If any minor updating to the source programs is required, this capability is also available through the VP-Routine.

Parameters are used as input to the data translators in order to direct the flow of execution. The three categories of functions which may be performed are data translation, data verification, and data validation.

Four modes of translation are available. The mode required

```
                 MOVE ALPHNUMERIC-D TO ALPHNUMERIC-X.
                 MOVE UNSIGNED-D TO UNSIGNED-X.
                 IF  SIGNED-D NEGATIVE
(MDD to SDF          MOVE "−" TO SIGNED-S
procedure)           ELSE
                     MOVE "+" TO SIGNED-S.
                 MOVE SIGNED-D TO SIGNED-X.

                 MOVE ALPHNUMERIC-X TO ALPHNUMERIC-A.
                 MOVE UNSIGNED-X TO UNSIGNED-A.
(SDF to MAD      MOVE SIGNED-X to SIGNED-A.
procedure)       IF  SIGNED-S EQUAL TO "−"
                     MULTIPLY −1 BY SIGNED-A.
```

Figure 2—Example of data translation procedures used in a Data Translator Program

```
                        IF  UNSIGNED-X NUMERIC NEXT SENTENCE ELSE
                            MOVE  UNSIGNED-X TO PRT-FIELD-DATA
(Data Validation            MOVE  "UNSIGNED-X" TO PRT-FIELD-NAME
 procedure)                 PERFORM PRINT-VALIDATION-ERROR.

                        IF  UNSIGNED-A NOT EQUAL TO UNSIGNED-D
                            MOVE  UNSIGNED-A TO FLDA-NUMERIC-18V
                            MOVE  UNSIGNED-A TO FLDA-NUMERIC-V18
                            MOVE  "UNSIGNED-A" TO FIELD-NAME
(Data Verification          MOVE  UNSIGNED-D TO FLDB-NUMERIC-18V
 procedure)                 MOVE  UNSIGNED-D TO FLDB-NUMERIC-V18
                            PERFORM  PRINT-VERIFICATION-ERROR.
```

Figure 3—Example of data validation and data verification procedures used in a data translator program

is indicated by the following parameter cards:

```
CONVERT MDD-SDF
CONVERT SDF-MAD
CONVERT MDD-MAD
CONVERT MAD-SDF
```

The first mode is applicable to the native compiler, and translates Machine Dependent Data to Standard Data Format. The second mode is applicable to the target compiler, and translates Standard Data Format to Machine ANSI Data. The last two data translation modes are used to create files for data verification.

There are three modes of data verification. The specific one required is indicated by one of the following parameter cards:

```
COMPARE MDD-MAD
COMPARE MAD-MADT
COMPARE SDF-MAD
```

The first mode of data verification is used to compare Machine ANSI Data files to Machine Dependent Data files. The second mode is used to compare two Machine ANSI Data files. The last mode is mainly for flexibility, and performs a SDF to MAD translation before a MAD to MADT comparison is made.

The third function performed by the conversion system is data validation. This consists of verifying that the data content is consistent with its class characteristics (i.e., numerically described fields should contain only numeric data). This function is performed automatically in combination with the translation function, or during a separate pass, using a VALIDATE SDF-DATA parameter.

Data validation/verification

In order to perform a comparative evaluation of the performance of computer systems through the use of natural benchmarks, we must ensure that the same amount of processing was completed by all competing systems, and that the accuracy of computations is within allowable bounds. Data comparison and validation procedures are included in our system for this purpose. Additionally, these procedures provide the benchmark recipient with a tool to check the various stages of a multi-step processing application for any processing inconsistencies. Finally, the procedures are used to confirm that data integrity is not lost in either the program conversion or data conversion process.

Processing integrity is verified in two ways through the authentication of data files and comparisons of files after program execution. The authentication of data files consists of validating the data item content for conformance to their described data class (i.e., numeric fields contain the data values Ø through 9 and, possibly, a sign). This specific feature was incorporated in the system because it has been found, for example, that some compiler implementations permit spaces as data in numeric fields, or maintain signed data in fields described as unsigned, and provide the appropriate translation before processing; other implementations do not. Validation would point out these potential problem areas.

The comparison of files after program execution provides a means of determining not only that all the processing was done, but also that the numerical results of this processing are within the accuracy limits allowed. When any data discrepancy is found by the data translators, a report of the discrepancy is produced. A report is made for each field in error, and includes the name of the field as defined in the program, its data content (in the case of a comparison, the field being compared to and the comparing field are both displayed), the position of the field in the record, and relative record position in the file. Record and error counts are also provided in the report. Figure 4 gives an example of the report generated.

PROGRAM MONITOR

One of the principal concerns in using benchmarks as a means of evaluating computer performance is whether they provide an adequate representation of the user's workload, and whether they properly reflect his future processing needs. This problem is not completely resolvable, but an indication

```
                         DATA VALIDATION/VERIFICATION REPORT
LOGICAL     FIELD          STARTING  FIELD
RECORD      NAME           POSITION  SIZE                    DATA CONTENTS
-------------------------------------------------------------------------------------------
000006   UNSIGNED-A          0021    0006   +0000000000000000443.000000000000000000 (INCORRECT)
                                                               *
                                            +0000000000000000444.000000000000000000 (CORRECT)
000020   ALPHNUMERIC-A       0001    0020   aaaaaaaacdeaaagjaaaa
                                                    ***    **
                                            aaaaaaaaaaaaaaaaaaaa
MACHINE ANSI RECORDS=004562
MACHINE DEPENDENT RECORDS=004562
ERROR COUNT=0002
```

Figure 4—Sample validation/verification report from a data translator program

of the processing characteristics of the programs provided for the benchmark can be of value in the evaluation process. This data is obtained through a program execution monitor.

Following program and data conversion, and before distribution of the benchmark to the vendors, this monitor is applied to the benchmark programs. The monitor, written in COBOL, inserts control statements into the benchmark programs. Execution of the benchmark programs with a given set of data provides a histogram of procedure activity in the programs. This, in turn, can be used to determine the suitability of the benchmarks in representing the user workload.

PACKAGING AND DISTRIBUTION

Once the benchmark has been sanitized and run on the native system to be assured that processing integrity has been maintained, the benchmark package is prepared for distribution. The package includes a source program library, benchmark data, and documentation.

The source library (population file) will contain the benchmark programs, data translator programs, the VP-Routine, and the operating system control language for the major computer systems. The VP-Routine selects the programs from the population file, transforms VP-Routine sensitive programs to machine dependent programs by satisfying all implementor defined names in the source program, and prepares the job control stream for submission to the operating system.

Data files for the benchmarks are distributed to the vendor on magnetic tape, in SDF format.

Documentation pertaining to the programs, data, instructions for implementation on a users system, and all information necessary to run the benchmark for a live test demonstration is included in the package. This includes the following information:

(1) Cross reference to data files by reel number.
(2) Cross reference to data files by program.
(3) Cross reference to data files for file name.
(4) Detailed instructions for implementation of the Data Translator/Verification Programs.
(5) Instructions for use of the VP-Routine.[3]
(6) A workload processing statement, which is a table providing a summary of all the pertinent information for implementation of the benchmark.
(7) Instructions and sample program for the extraction of the VP-Routine from the population file.
(8) Benchmark instructions.
(9) Individual program documentation, including any known areas which may cause implementation problems.
(10) For variable length records, or multiply defined records, a complete COBOL record description is given, or a record layout is provided together with its record type characteristics.
(11) A system flowchart of the benchmark.
(12) Listings of each program.

LIMITATIONS

Even though the Benchmark Preparation System resolves many of the difficulties involved in program and data portability, there are areas in which reprogramming will be required for complete conversion. The amount of reprogramming depends on the degree to which machine dependency has been imposed onto the program. Data that is not explicitly defined, or features for which ANSI Standard COBOL does not have a direct functional replacement cannot be detected by the sanitation process. The following are a few of the known programming, COBOL characteristics, or COBOL compiler implementation practices which have an impact on automation of the conversion process.

(1) *Functions in the native COBOL source program which cannot be directly replaced by features or elements of the ANSI language specification.* Such an example would be the READY TRACE statement or the TRANSFORM verb in IBM System/360 COBOL.[5] The ANSI language specification does not have an element or feature which directly performs these functions. To simulate this function requires manual conversion.

(2) *Incomplete or inadequate record descriptions.* This is due to describing fields or groups of fields as alphanumeric when their true descriptions could include other forms of data representation. An example of this technique on the IBM 360/370 would be a data field described as PICTURE X(4), when the data actually present should be defined as PIC S9(9) COMPUTATIONAL (binary), or a PIC X(2) definition of a data field which is in fact PIC S9(3) COMPUTATIONAL-3 (packed decimal). The above examples would not only cause the target compiler to incorrectly allocate storage but also would not provide the appropriate conversion processing, since the data is described as alphanumeric.

(3) *Multiply defined records which have different data structures within each record, and do not have a means of distinguishing between records.* The data conversion process is capable of translating multiply defined records, but only if they can be identified.

(4) *Machine dependencies fixed into the COBOL program itself.* This would include such things as assuming the initial value of a data item, initializing numeric storage areas with alphanumeric literals representing a machine's internal sign, or using the character set to represent non-character machine data. An example would be:

```
WORKING-STORAGE SECTION.
77 SIGN-FIELD PIC S999.
77 X-FIELD REDEFINES SIGN-FIELD
   PIC XXX.
PROCEDURE DIVISION.
SECTION-NAME SECTION.
PARAGRAPH-L.
MOVE +123 to SIGN-FIELD
IF X-FIELD EQUAL TO '12C' GO TO—
```

Implementations which do not generate positive sign over-punches would require the procedure to be modified before the program would function correctly.

(5) *Collating sequence of fields containing alphanumeric data which are critical to program processing and which are not completely defined.* This problem is somewhat reduced, however, in that COBOL instructions which may be affected by collating sequence are flagged by the COBOL to COBOL translator.

CONCLUSIONS

The Benchmark Preparation System was developed to reduce the nonportability and expense of using natural benchmarks without losing the characteristics of the users workload in terms of processing efficiency and representation. The results we have obtained indicate that these objectives can be met.

Our current test bed is a Navy benchmark containing 38 COBOL programs consisting of some 60,000 lines of source code, and includes some 150 data files. The native system is an IBM 360/50 and the target machines are a UNIVAC 1108 and HIS 6050. These programs and data files have been successfully converted to both the UNIVAC 1108 and HIS 6050. Preparation of the benchmark programs, development of data translator/verification programs, and the packaging of these were done on a UNIVAC 1108. Approximately 96 percent of the changes made to the programs were handled by this system. The remaining changes (manual) were necessitated by extension features with no counterparts in the ANSI COBOL standard. Generation of the data translators and sanitation of the benchmark programs for packaging required approximately two computer runs and one man hour of effort per benchmark program. The effort required on each system to set up the VP-Routine, and cleanly compile the programs has been averaging one-tenth man hour per program. Character code translation posed no problem, as each system had job control card options for transliteration (i.e., EBCDIC to BCD on the IBM/360 and BCD to FIELDATA on the UNIVAC 1108 and IBMC code to HIS 6000 code).

Based on our efforts, we believe that portability can be achieved by an automated means without sacrificing the efficiency of a computer system.

REFERENCES

1. Department of the Navy, *Specification, Selection and Acquisition of Automatic Data Processing Equipment (ADPE)*, SECNAVINST 5236.1, December 17, 1971.
2. Ickes, Hubert F. and Herbert S. Meltzer, *Draft Tutorial on Interchangeable Data Files*," ANSI Task Group X3.2F (1970).
3. Chief of Naval Operations, Information Systems Division (Op-91), *COBOL Compiler Validation System*, VP-Routine users guide, January 1973.
4. American National Standards Institute, Incorporated, *USA Standard COBOL*, X3.23-1968.
5. *IBM System/360 Operating System Full American National Standard COBOL.* GC 28-6396-2, IBM Corporation (1970).

An experiment in the use of synthetic programs for system benchmarking

by PAUL OLIVER, GEORGE BAIRD, MARGARET COOK, ARNOLD JOHNSON and PATRICK HOYT

Department of the Navy
Washington, D.C.

BACKGROUND

Competitive computer system selection requires a tool for minimum performance measurement. The selection process must be fair and, ideally, brief and economical. Thus, the measurement tool must be visibly fair and impartial in its measurement of a computer system, it must relate what is being measured to user needs, and it must be economical to apply. The thrust of several ongoing "standard benchmark" efforts in the Department of Defense and other Federal Government agencies is to develop a measurement tool with these qualities.

There are several characteristics of computer systems which can be measured for the purpose of selection:

(a) Availability of equipment and software, in terms of reliability, maintenance time, and the like.

(b) Work capacity, which can be measured from a variety of viewpoints. *Job time* is a single-job measure and, therefore, not often used. *System throughput* is a measure of how much work is done, and is a function of the job mix and job load, as well as various system parameters. *Response time* is a measure of the quality of service rendered, and is largely dependent on operating system and hardware characteristics.

(c) Functional capabilities are susceptible to qualitative judgments, but demonstrations of these capabilities are often required of computer system vendors (e.g., a demonstration of an on-line text editor).

In the context of computer selection, we have felt it prudent to limit the scope of our efforts to measuring throughput capacity, recognizing, however, that the other factors may take on paramount importance under varying circumstances.

Relation to performance evaluation

It is important that we recognize the affinity of any benchmark study to the subject of computer performance evaluation, since some combination of evaluation techniques will of necessity be used in the development of "standard benchmarks." These techniques can be broadly classified and characterized as follows:[1]

(a) *Task-oriented* techniques concern themselves with system throughput capabilities with respect to a given workload. Simple instruction timings reduce the "workload" to specific classes of instructions (add time, floating-point multiply, etc.). Instruction mixes consist of "representative" samples of instruction sets designed to reflect the degree to which each instruction class is used for a given type of application. These are adequate for estimating *processor* power, but completely ignore memory, degree of multiprogramming, I/O loads, etc. Kernels are relatively small sequences of code performing a single (simple) function (e.g., a table search), and, again, are designed primarily for measuring processing power. The timings for kernels may be obtained by actually executing them or by hand-calculations. Benchmarks consist of a subset of a given workload ("natural" benchmarks), a subset which has been further modified ("hybrid" benchmarks), or a set of programs written specifically for the purpose of making a comparative evaluation ("synthetic" programs). Benchmarks are processed on the configurations being evaluated or compared, and the processing time is used as a relative figure of merit.

(b) The emphasis in *component-oriented* evaluation techniques is on the system being evaluated rather than on the workload to be processed by this system. Hardware monitors are relatively inexpensive, precise in what they measure, non-disruptive, but insensitive to data-dependent information. The characteristics of software monitors are almost the precise opposite of those for hardware monitors. The convenience of queueing models is offset by their inaccuracy and shallowness. Stochastic models (simulation models) are less imprecise but costly, and suffer from a credibility gap.

Problems with natural or hybrid benchmarks

Benchmarks have for some period of time constituted the accepted form of minimum performance measurement in computer selection throughout the Federal marketplace. Natural or hybrid benchmarks have the advantages of dealing

```
                    X-30     PRINTER
                    ...
INPUTS              X-65     UNIVAC-1108

                    X-66     UNIVAC-1108
                    ...

                    SOURCE COMPUTER.
                      XXXXX65.

POPULATION          OBJECT COMPUTER.
FILE                  XXXXX66.
FORMAT
                    ...

                    FILE CONTROL.  SELECT RESULTS ASSIGN TO
                      XXXXX30.

                    ...

                    SOURCE COMPUTER.
COMPILATION           UNIVAC-1108.
TIME
FORMAT              OBJECT COMPUTER.
                      UNIVAC-1108.

                    ...

                    FILE CONTROL.  SELECT RESULTS ASSIGN TO PRINTER.
```

Figure 1—Example of VP-Routine input, population file form of audit routines, and compilation-time form of audit routines

with a real system (thus avoiding half of the simulation credibility problem) and a "semi-real" job mix. Among the more serious problems associated with benchmarks are the following:

(a) It is extremely difficult, except in the simplest situations, to construct a set of benchmark programs which accurately reflects a given job mix. This of course is a problem common to any performance measurement technique, since the nature of "a given job mix" is dependent on a multitude of parameters, many of which are system dependent (e.g., *EX*ecute *C*hannel *P*rogram instruction counts are often used to measure I/O time on IBM S/360 or S/370 systems, but these instructions have little meaning outside the S/360-370 series, and often have no precise counterparts on other systems) and most of which are time dependent.

(b) They are generally non-portable (system dependent) and often do not run correctly, even on their native system.

(c) They are prepared and processed using a variety of procedures resulting in unduly long execution times, unreasonable file volumes, and inconsistent measurement procedures. This author has seen benchmarks for which the required processing time was better than three hours, and the file population resided on two dozen (full) tape reels! In some cases only processor time is measured; in others, all components (including, e.g., printers) must halt before timing stops.

(d) The above problems result in extremely high costs, to buyers and vendors, in terms of both time and money. It is not unusual for a vendor to spend 6-9 calendar months just to prepare the submitted benchmarks for processing, or for the cost of processing them to be 10 percent or more of the eventual bid price.

SCOPE OF THE U. S. NAVY EXPERIMENT

The Software Development Division of the Department of the Navy Automatic Data Processing Equipment Selection Office (ADPESO) is performing an experiment to determine the suitability of synthetic programs in alleviating the problems created by natural and hybrid benchmarks.

The experiment began in June 1973, with the development of a small (5 program) reference library of synthetic programs. We assumed that synthetic programs could be written so that relatively few parameters control their behavior; experimentation could be performed on these programs so that their behavior relative to changing parameter values would be predictable; specifications of a workload *based on the parameters implicitly defined by the synthetic programs* could be made, and synthetic program parameters could be set so as to reflect this workload.

The use of synthetic programs in performance evaluation does not represent a new concept. Dopping,[2] and Gosden and Sisson[3] reported on experiments in the use of synthetic programs as far back as 1962. More recent suggestions on their use have come from Joslin[4] and Buchholtz.[5] Our aims have been to obtain quantitative profiles of certain synthetic programs and to determine the scope of their feasible utility.

RELATED EFFORTS

There are several complementary efforts in the Federal Government aimed at designing representative benchmarks.

The U. S. Army Computer System Support and Evaluation Command has recently issued a solicitation for a "Standard Benchmark Study." The contract objectives are (a) The definition of all tasks and measurable functions performed by a computer in executing business-type applications; (b) Development of a method or technique of identifying and measuring the occurrence of each function or parameter in

```
PROJECT:  SYNTHETIC BENCHMARKS

MODULE:  SEQUENTIAL I/O

COMPILE TIME PARAMETERS:

     1.  Records/Block - for all files; impacts buffering.

     2.  Record Size - for all files and to reflect application.

     3.  Start Variable - used to vary accuracy requirement in compute
         kernel.

     4.  Table Size - to impact memory requirements.

     5.  Data Types - to reflect application.

EXECUTE TIME PARAMETERS:

     1.  Master File Size - to impact i/o time

     2.  Detail File Size - in conjunction with "repetitions" can impact
         processing time.

     3.  Repetitions - number of repetitions of a compute kernel per
         master-detail watch.


NOTE:  See listing for more details.
```

Figure 2—Sequential I/O module parameters

each task for the purpose of profiling computer workloads. This solicitation is the result of a careful study on the part of a Department of Defense Joint Steering Committee which has, among other things, defined a preliminary set of application tasks and task parameters for benchmark purposes.

The Department of Agriculture has constructed a comprehensive set of benchmark programs which include transaction processing and data base management applications. There is much in this package which should be carefully studied as part of any effort at designing a library of standard benchmark programs.

The Department of Labor is developing a job selection simulation model[6] using actual utilization statistics as control parameters. Although the goals here are somewhat different from those of the "standard benchmark effort" there may be some related spinoff benefits.

A similar project is being carred on by Marine Corps using hardware monitors to provide data for the synthetic creation of jobs.[7]

RESULTS

The programs

Five processing tasks were selected as representing, in varying combinations, a broad variety of application tasks. These were sequential file processing, indexed sequential file processing, relative I/O processing, sorting, and computation.

Programs were written to perform each of these tasks. Because most of the Navy's present benchmark needs relate to COBOL-oriented workloads, all of the reference library programs are written in American National Standard COBOL. Additionally, all the programs are in "system independent" form. This is accomplished through the use of an executive program, the *VP-Routine.* The VP-Routine was developed in 1969 by the Department of the Navy as part of its *COBOL Compiler Validation System.*[8] It is used to resolve implementor names (e.g., in the ENVIRONMENT DIVISION), modify compile-time parameters (e.g., record sizes, precision, blocking factors), and automatically generate job control instructions appropriate to the system we are executing under (Figure 1).

Each program is controlled by a set of compile time and execution time parameters. Figures 2-6 identify these for each of the five programs. The ability to vary automatically certain parameters at compile time provides us with the flexibility to develop a fairly rich mix from just a few basic programs.

We have adopted certain design principles which, while applicable to software design in general, we felt were particularly important to this project.

(a) We have attempted to make every detail of the structure of each program visible and understandable to a prospective user. This is a prerequisite to a "sellable" product.

(b) The design of each program is consistent with that of the others. We have used "modular programming" throughout, although, frankly, this was simply a reflection of following long accepted standards of good programming practice. We maintained consistency in the binding time of parameters across programs. Thus, if a given parameter is bound at compile time in one program it is bound at compile time in all the programs. Also, all files used by a program are generated by that program (eventually, the file generation modules may be combined into one program).

(c) We have isolated the function of each of the program parameters so as to render each parameter independent of

```
PROJECT:  SYNTHETIC BENCHMARKS

MODULE:  INDEXED SEQUENTIAL UPDATE

COMPILE TIME PARAMETERS:

    1.  Memory Variable - is set by adjusting the size of a table in
        working-storage.  This is available to vary the memory storage
        requirement of the program.

    2.  Record size - Default is 800 characters.

    3.  Block size - Default is 10.

    4.  Index key size - Default is 10.

EXECUTE TIME PARAMETERS

    1.  Master File Size - sets the number of records to be created for
        the master file.

    2.  Detail File Size - sets the number of transactions to be processed
        against the master file to measure I-O processing.

        (a) Deletion Percent - is percent of detail transactions which
            initiate deletion of master records (default is 10 percent).
            This parameter is available to measure the affect of record
            deletion type transactions on I-O processing time.

        (b) Addition Percent - is percent of detail transactions which
            add records to the master file (default is 10 percent).  This
            parameter is available to measure the affect of transaction
            insertion into the index file on I-O processing time.

        (c) Sequential Percent - percent of detail transaction which initiate
            processing the index file sequentially (default is 5 percent).
            This is to measure the affect on I-O processing when accessing
            the index file sequentially.

    3.  Computation Repetitions - sets the number of times the program
        cycles through compute bound procedurds.  This parameter is
        available to place a workload on the CPU.
```

Figure 3—ISAM module parameters

```
PROJECT:  SYNTHETIC BENCHMARKS

MODULE:  RELATIVE I/O

COMPILE TIME PARAMETERS:

    1.  Master and Detail Files Record Size - minimum of 120 characters -
        the user can request a larger record.

    2.  Master and Detail Files Block Size - minimum of 1 record per block -
        the user may request a larger block size if applicable.

EXECUTE TIME PARAMETERS:

    1.  Number of Master Records -(5,000 default) - the user could request a
        larger or smaller number of records.

    2.  Order the Records are created - (sequential default) - The default
        causes the files to be created with 10% "missing" records, i.e. 5,000
        records from 1 to 5,500.  The user may request that a different per-
        centage of gaps be left between records for inserting purposes.

    3.  Number of Detail Records - (2,500 default)

    4.  Percent of the Following - (100% total):

        (a) Detail records which match master records and cause an update
            to take place.  (33% default).
        (b) Detail records which match master records and cause the master
            record to be deleted.
        (c) Detail records which do not match master records and cause a
            new master record to be created
```

Figure 4—Relative I/O module parameters

PROJECT: SYNTHETIC BENCHMARKS

MODULE: SORT

COMPILE TIME PARAMETERS:

1. Record Length - Used to impact
   (a) Buffer size.
   (b) Transfer time.
   (c) Internal and external storage requirements
   (d) Whether minimum and maximum logical size of applications can be handled.
   (e) Whether sort can handle variable length logical records.
2. Blocking Factors - Used to affect
   (a) Buffer size.
   (b) Transfer time.
   (c) Ratio of inter-record gaps/data for magnetic tape; hence external storage requirements.
   (d) Mass storage partition use/waste ration; hence mass storage requirements and number of seeks and transfers required.
   (e) Whether minimum and maximum physical record size can be handled.
   (f) Whether padding is required.
   (g) Whether extra characters must be added to each physical record if the file is blocked.
   (h) Provides a way to increase I/O time used for a single transfer to change I/O to computer ratio.
3. Number of Sort Keys - Affects
   (a) Number of sort passes required to produce specified sequence.
   (b) Test that the number of keys allowed in a single sort step equals these required by an application.
   (c) Total length of sort field.
4. Type of Sort Keys - Determines
   (a) Whether all types of keys required by an application can be handled (numeric, alphabetic, alphanumeric, signed, decimal points).
   (b) Time required for various types of comparisons, numeric vs. alphanumeric.
   (c) Points out the collating sequence used by the machine for sorts and compares.
5. Order of Sort Keys - Prevent cheating by setting at test time to compare results against predicted behavior of final sort sequence.
6. Ascending or descending sort -

EXECUTE TIME PARAMETERS:

1. Number of Records -
   (a) Total data volume for input.
   (b) Whether sort can be done completely in core.
   (c) Amount of dependence on mass storage for intermediate merge strings.
2. Number of Computations on I/O -
   (a) Ability to simulate amount of modification done during sort process.
   (b) Changes ratio of added computer requirements/sort I/O processing.

NOTE: Although not specifically specified as a compile time parameter, the file assignments for INPUT-FILE, SORT-FILE, and OUTPUT-FILE can change the basic sort characteristics from mass storage to tape orientation. This affects file rewind time, transfer rates, and blocking conventions.

Figure 5—SORT module parameters

PROJECT: SYNTHETIC BENCHMARKS

MODULE: COMPUTE

COMPILE TIME PARAMETERS:

1. Table Size - used to vary the size of an in-core table, thus allowing for modification of memory requirements.
2. Data Descriptions - modified by appropriate changes to respective PICTURE clauses. Used to vary computation accuracy requirements and processing time.

EXECUTE TIME PARAMETERS:

1. Constants - for random number generator
2. Processing Iterations - to vary CPU activity.
3. Accuracy Parameter - used to vary accuracy requirements.
4. Processing Deletion Switches - to indicate coding to be s ipped.

NOTE: All parameters have default values — see program listing for details.

Figure 6—Compute module parameters

the others. This was necessary to avoid facing an exponentially rising set of options in setting parameters to control program behavior. This was a difficult principle to follow since, for example, a simple specification such as how one is to control I/O time can be made in terms of file size, blocking factor, logical record size, etc. In this case we could choose to use file size to effect time, blocking factor to impact buffering, and maintain logical record size constant.

(d) Only those functions which were felt essential to the accurate modeling of a task were included in each program. Thus we opted for a clearly defined scope and simplicity rather than complexity. We feel this was particularly important in the selection of synthetic program functions and parameters, since a lack of frugality can lead to a level of complexity in the programs which would have rendered them completely unamenable to analysis.

(e) The design of each program (and of the set of programs as a whole) lends itself to extension, so that a wide range of task characteristics can be accommodated.

Each program is self-documented. A "prologue" is included for each and commenting is plentiful, though pertinent. External documentation consists of a "module overview" (see Figure 7), parameter specifications, experimental results, and a User Guide to assist an organization in implementing the programs and using the VP-Routine. We have avoided lengthy descriptions and detailed flowcharts because we question their usefulness.

PROJECT: SYNTHETIC BENCHMARKS

SEQUENTIAL MODULE OVERVIEW

PROGRAM-ID: SEQPRGRM

PURPOSE:

This synthetic program is designed to reflect the properties of a sequential file update process.

USAGE:

In its machine independent form SEQPRGRM is designed to be used in conjunction with the VP-routine (see references). In machine dependent form, SEQPRGRM is a stand-alone program.

ENVIRONMENT:

This program was developed on a UNIVAC-1100 System. It is designed to function correctly when translated by a COBOL compiler conforming to Federal COBOL standards as interpreted by the COBOL Compiler Validation System.

METHOD:

Master and detail sequential files are created, together with an in-core table. Timing for this program is then initiated. The master file is compared against the detail file until a key match is made. For each occurrence of a key match an update of the master file is made (creating a new master file), and a compute kernel is executed a varying number of times. When the detail file is exhausted,timing for this program is terminated and a summery record is written.

REFERENCES:

Navy COBOL Compiler Validation System User Guide
  Information Systems Division (Op-91)
A synthetic job...Bucknoltz, IBM Sys. J. (4), 1969

Figure 7—Example of a synthetic module overview

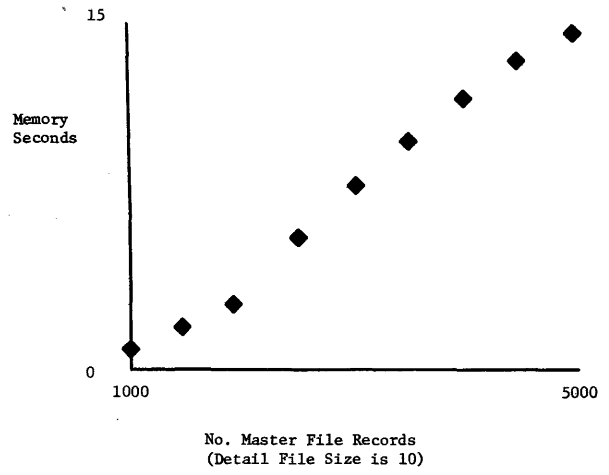


Figure 8—Sequential file update time as a function of master file size—no CPU activity, drum-resident files

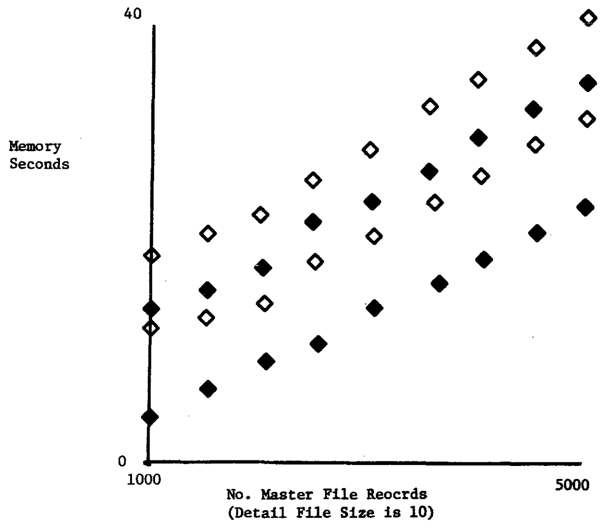


Figure 9—Sequential file update time as a function of master file size—no CPU activity, tape-resident files

The programs, documentation, and VP-Routine are collected on a 2400 foot magnetic tape reel. The User Guide and experimental results on program behavior are separately bound. The entire package is in the public domain.

Examples of processing results

A complete summary of processing results is beyond the scope of this paper, but we can discuss some of the more interesting of those results. All results mentioned are based on executions on a UNIVAC 1108 Unit Processor, under control of the EXEC-8 Operating System.

The "sequential I/O" module is the simplest of the file processing programs. Its function is to pass a master file against a detail file, creating a new master file. The files may reside on tape or direct access devices. A compute loop may be performed a variable number of times each time a master file record is updated. The processing includes a table search, and the size of the table is used to control memory requirements. All computations are self-checking. The program is similar in these and other characteristics to the PL/1 program described by Buchholz.[5]

Predictably, we found I/O time to be a linear function of master file size. This was true for FASTRAND (drum) resident as well as tape resident files. Repeated runs during different times of day showed that the curve reflecting the behavior of time as a function of master file size remained a straight line with constant slope, although the intercept value changed (Figure 8). In all these runs, only the master file size was varied (from 100 to 5000 records), with the detail file size fixed at 10 records), and only one pass through the compute loop was performed.

We processed a series of similar runs with all files residing on UNIVAC 8-C tapes. Again, running the program in a mix did not change the linear behavior of time as a function of file size (Figure 9). As before, the detail file size was held constant, and only one pass through the compute loop was performed on each record update. Thus, while other programs in a mix clearly affect the *quantitative* behavior of a sequential update task, they appear to have almost no effect on its *qualitative* behavior.

CPU time turned out to be a linear function of the number of repetitions through the compute loop.

Execution of the "compute" module produced some interesting results. The program generates a variable-sized table of uniformly distributed pseudo-random numbers, performs a "runs-up-and-down" test on them, and optionally produces printer output. A parameter controlling the number of processing iterations is used to vary the amount of CPU activity.

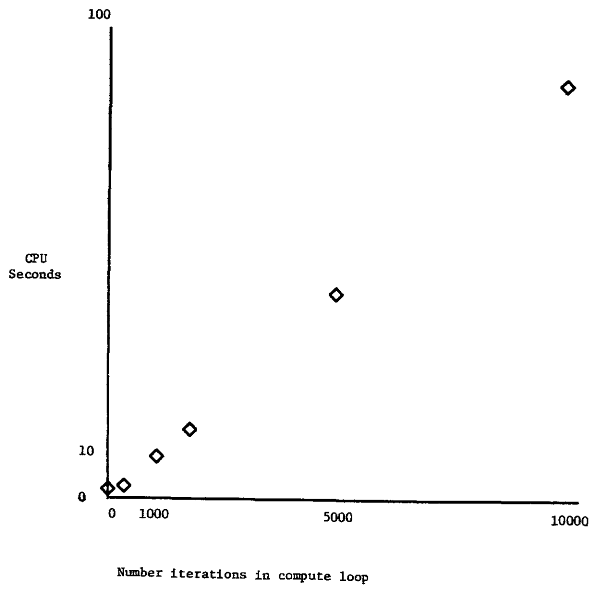


Figure 10—Compute module CPU utilization as a function of number of iterations in the computation loop

Number of Iterations	CPU Time (minutes) (Display Mode	CPU Time (minutes) (Computational Mode)
20	.083	.643
100	.816	.007
200	1.497	.515
500	4.525	1.906
1,000	9.531	2.990
1,700	14.945	5.062
5,000	45.524	14.156
10,000	89.941	23.324
20,000	158.507	47.696

Figure 11—Compute module CPU time utilization as a function of number of iterations in compute loop

When the number of iterations reached a certain threshold (usually 500) the CPU time varied linearly with this parameter. Below that point, however, we noticed some fluctuations (Figure 10). We believe this is due to the way the EXEC-8 dispatcher schedules jobs for CPU time. (It uses a variation of Corbato's time quantum charging algorithm.[9])

Figure 11 summarizes two executions, run under identical conditions. The only difference was that in one the usage of variables was "computational," in the other "display." As a program becomes CPU bound an exorbitant price is paid for the "machine independency" of data.

Figure 12 shows the relationship between memory time (for a given program, a memory second is defined as the occupation of 32K words of memory for a period of one second, during which time the program is undergoing either CPU or I/O activity) and the size of the file being sorted for the "sort" module. Again, we found a linear behavior, and this pattern was consistent regardless of other jobs in the mix, time of day, etc. Fluctuations at the low end of the line were due, as in other cases, to EXEC-8 allocation characteristics.

Problems encountered

We feel confident, based on our tests thus far, that we can indeed modify program parameters, for the modules we have produced, in such a way that we can force a predictable behavior on the programs, in terms of both time and pattern. This, however, only tells us that we can control the programs —a necessary but not sufficient condition if we are to create synthetic benchmarks.

We have also encountered certain difficulties with the synthetic program approach. Not all of these are unique to this approach, but this offers us little solace. The following were the most serious of these problems:

(a) Because synthetic programs tend to be stylized, they may produce surprising results. For example, an optimizing compiler can have a much greater impact on a synthetic benchmark than on a natural one. Yet, user workloads are "natural," not synthetic. We have found that PERFORM sections which are called only once, and not otherwise entered, are placed in-line by many compilers, but not by all. This creates no difficulties if a user creating a set of benchmarks knows what his compiler does, but he does not have to know. Also, sequences of code such as

$$I=I+1$$

$$A=I,$$

where I is a loop-control parameter (the syntax here is FORTRAN but the principle is equally true of COBOL) are generally not performed as such by an even moderately intelligent compiler.

(b) Another problem we have encountered is that overwhelming side effects can occur in overly parameterized synthetic programs. For example, the COBOL PERFORM verb translates to 14 instructions on one system we executed under, while the MOVE verb translates to 1 instruction. Thus, using the PERFORM instruction to vary the number of times a MOVE instruction is executed leads to grossly misleading results when the PERFORM itself is the object of yet another PERFORM.

(c) One needs to understand the "native" system in some detail in order to develop benchmarks purporting to accurately reflect a given workload for that system. Some of the test results cited above, for example, were clearly due to the nature of the system on which the programs were executed. This means that guidelines on how to use the synthetic modules will differ with differing systems. Also, it is easy to create an unduly complex program (in terms of possible combinations of parameters) if the architecture of the native system is not understood. Repeating, for instance, a series of COBOL MOVE's, varying field sizes each time, accomplishes nothing more than what could be accomplished by moving a fixed size variable on IBM S/360 computers, since a single machine instruction, MVC (move character) is used regardless of field size. Yet, on a UNIVAC 1108, changes in object code

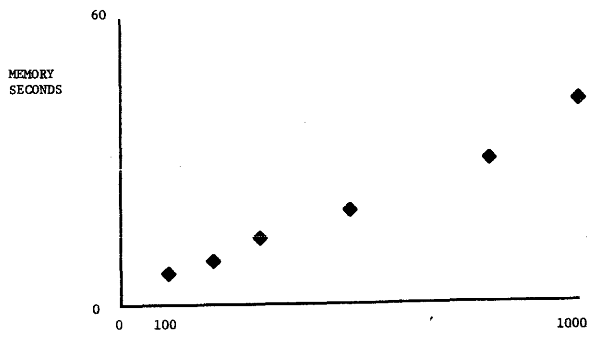


Figure 12—Sort module memory seconds utilization as a function of number of records sorted

do occur at certain field sizes. Also, moves of literals, numerics, and character fields are usually all performed in the same way, so that incorporating all of these in a program is simply adding to the combinations of parameters without really contributing to the value of the program.

(d) We see no evidence of a satisfactory way of modeling a workload. Even a simple I/O—CPU analysis of a file maintenance problem depends on a multitude of parameters: proportion of active to passive records, distribution and location of active records in the master file, number of instructions executed per active/inactive record, record size, frequencies with which instructions are executed, etc. This difficulty is seriously aggravated in a *mix* of programs. It is not at all clear that techniques for matching job parameters to mix parameters is feasible. The use of analytical models to characterize a job mix and thereby provide inputs to the synthetic programs[1] is clearly unsatisfactory, since the limiting factor would then become the analytical techniques themselves. This class of techniques is already regarded as grossly imprecise.

The use of software monitors for data collection is likewise unacceptable since they create serious instances of the "Hawthorne" effect.[10] This could possibly be compensated for, but with considerable difficulty.

In fact, it is important to note that *all* suggestions on how to model a workload rely on one of the evaluation techniques previously surveyed (monitors, simulation, etc.). Thus, we should not expect the synthetic mix approach to be an improvement over these.

The problem of "representativeness" which exists in natural benchmarks will simply not disappear just because we use synthetic programs. We have cited the system dependency of workload parameters (particularly as they apply to I/O time) and the sheer magnitude of the number of combinations of program parameter values. An equally crucial problem is the fact that the nature of a workload is time dependent. Any attempt to condense a workload into a, say, two-hour benchmark is bound to result in substantial homogenization, and some important characteristics could be lost. As a simple example, the annual workload of a computer center, in terms of productive hours, is given in Figure 13. It suggests that there is plenty of excess capacity. Yet the workload on a typical mid-week day shown in Figure 14 indicates that for this period the system was saturated. We know of no satisfactory techniques which allow us to model this behavior for the purpose of building benchmarks.

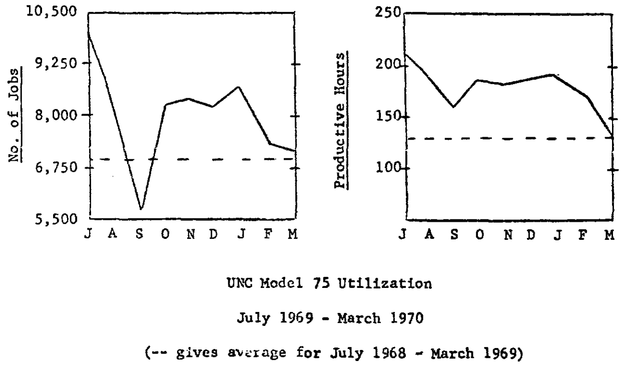


Figure 13—Monthly utilization profile (Source: Annual Report, University of North Carolina Computation Center, 1970)

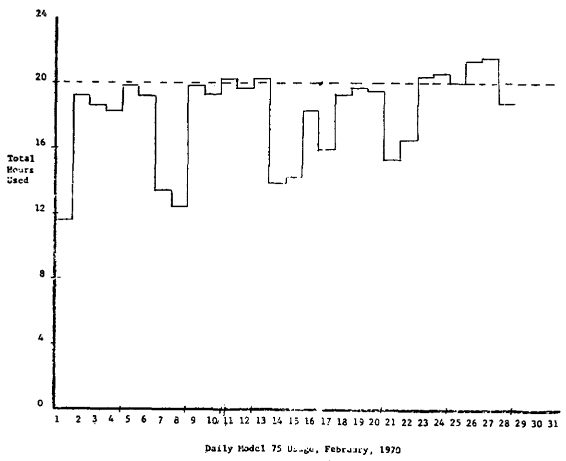


Figure 14—Daily utilization profile (Source: Annual Report, University of North Carolina Computation Center, 1970)

CONCLUSIONS

Can a controllable job mix be constructed?

We believe, on the basis of our experience thus far, that task-oriented synthetic programs can be combined into a mix which can be controlled to exhibit desired processing time, memory, I/O time, and I/O devices utilization characteristics. There have been other efforts that bear this out.[11] We plan additional testing on a variety of systems so as to learn more about some of the system dependencies we have encountered.

Can a workload be profiled?

We do not believe that it is possible to arrive at a generalized, comprehensive, and accurate model of system workloads except in the most trivial cases. We can certainly retrofit. That is, we can accept a workload definition based on the synthetic program parameters. We also believe that this need not impede the use of synthetic programs in benchmarks. In this, we strongly support the view expressed by J. C. Strauss. In a recent paper[12] on the use of natural benchmarks, he stated that, based in part on prior experience and on the difficulties encountered, "it was felt more important that the behavior of the benchmarks be well understood and cover a broad range of important system features than that the complete benchmark series be representative of the general workload."

Other uses for synthetic programs

Isolated system characteristics can be exercised using synthetic programs. We have in fact used the I/O modules in our reference set to test various operating systems data management capabilities. Synthetic programs also serve as convenient tools to determine the impact of certain programming practices, as was done in using the "compute" module to measure the degradation, on a specific system, resulting from COBOL DISPLAY mode computation.

A recommendation

We feel our testing has substantiated our original assumptions. A small number of simple, task-oriented, synthetic programs can be combined into a fairly rich and versatile job mix. A relatively small number of parameters is sufficient to enable a single program to reflect the characteristics of a broad class of applications. Also, individual modules have proven useful in exercising isolated computer system features, such as I/O handling. Finally, if one accepts a "modest" workload characterization, aimed more at reflecting extremities and crucial areas rather than comprehensiveness, it is possible and reasonable to construct a benchmark from a set of synthetic modules.

Synthetic programs are neither difficult nor expensive to produce. Our present set, admittedly small, was designed, coded, and debugged in two calendar months. An additional three months were required for experimentation, packaging, and system documentation. These times do not consider the VP-Routine, which was already available. Total manpower used for the effort amounted to four man-months. Total cost, including machine time, clerical support, and salaries was under $6,000. Furthermore, the system is available to anyone upon request. Thus, we feel we have made a small investment for a product which has already given a substantial payoff, in what we have learned if nothing else.

A reference set of "controllable" programs is a useful tool for any data processing installation. Our concern was primarily with benchmarks for system selection. We have indicated that performance measurement is a related area of application. System sizing, throughput estimates against a changing workload, expected response time to a varying stimulus, and availability measurements are other reasonable applications for a set of synthetic modules. The modesty of the effort required to produce such a set certainly commends further study.

REFERENCES

1. Lucas, H. C., "Performance Evaluation and Monitoring," *ACM Computing Surveys*, 3, 3, 1971.
2. Dopping, D., "Test Problems Used For the Evaluation of Computers," *Bit*, 2, 4, 1962.
3. Gosden, J. A. and R. L. Sisson, "Standardized Comparisons of Computer Performance," *Proc. 1962 IFIP Congress.*
4. Joslin, E. O., "Application Benchmarks: The Key to Meaningful Computer Evaluations," *Proc. 20th ACM Nat. Conf.*, pp. 27-37, 1965.
5. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM System Journal*, Vol. 8, No. 9, 1969.
6. Byrne, T. A., et al., "A Job Selection Simulation Model," *Symposium on the Simulation of Computer Systems* (ACM), June, 1973.
7. Hesser, W. A., "Creation of a Simulation Model From Hardware Monitor Data Using the SAM Language," *Symposium on the Simulation of Computer Systems* (AMC), June 1973.
8. Baird, G. N., "The DOD COBOL Compiler Validation System," *Proc. FJCC*, 1972.
9. *UNIVAC 1100 Series Operating System Programmer Reference*, UP-4144, Sperry-UNIVAC (1973).
10. Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement," *IEEE Computer Journal*, July/August, 1972.
11. Wood, David C. and Ernest H. Forman, "Throughput Measurement Using a Synthetic Job Stream," *Proc. 1971 FJCC*, AFIPS Press, Vol. 39.
12. Strauss, J. C., "A Benchmark Study," *Proc. 1972 FJCC*, AFIPS Press, Vol. 41, Part II.

A microprocessor implementation of a dedicated store–and–forward data communications system

by P. M. RUSSO and M. D. LIPPMAN

RCA Laboratories
Princeton, New Jersey

INTRODUCTION

The stored-program approach to data communications system design is not new. The past several years have witnessed a large and ever-increasing number of mini-computers and larger processors dedicated to the implementation of a variety of data communications functions. To date, however, the use of computers has been relegated primarily to medium-sized and larger systems where highly complex data communications requirements justify reasonably large investments in hardware and software. In many low-end applications, however, the high cost of minicomputers and their associated peripherals cannot be justified. This is especially true in a dedicated system where two terminals (or groups of terminals) communicate over a dedicated communications channel.

The advent of low-cost LSI microprocessors and mass storage devices (e.g., floppy discs) is having a significant impact on the design of new low-end data communication systems. A multitude of systems that, until recently, would have required a hard-wired logic implementation with logic speeds far in excess of the system requirements, can now realize the many advantages of the stored program approach. These advantages include, among others, lower cost, programmability (flexibility), improved reliability, ease of maintenance, and the addition of many new system functions hitherto impractical to implement via hard-wired logic. The many advantages of micro-processor implementations of data communications systems are discussed more fully below.

In this paper we will describe a dedicated store-and-forward system that may prove suitable for international data communications. The system is configured around the RCA COSMAC LSI microprocessor and the Century Data Systems CDS-110 floppy disc, with suitable disc, keyboard, display and communications interfaces. The system architecture, disc interface organization, COSMAC microprocessor, data structure and the system functional capability will be detailed. Emphasis will be placed on various new functions achievable with stored-program control. Finally, other potential applications of microprocessors in the data communication field will be briefly discussed.

LEASED CHANNEL SYSTEM

The dedicated store-and-forward data communications system that we have implemented is functionally related to currently commercially available international leased channel systems. Hence, for the purpose of this paper, we will also refer to the microprocessor based system as a leased channel system. An international leased channel is a dedicated communications link between a customer's domestic and foreign offices, Figure 1. Typically, this link is made via a leased channel control unit (CU) that performs a variety of functions. Many CU's are usually located in a single centralized control room. The CU acts as an interface between domestic and foreign communications networks. This includes both electrical interfacing and message format interfacing, such as code and speed conversion. Typically, the international link employs 5-level baudot character encoding whereas the domestic link uses 8-level ASCII. The control unit also handles character expansion, handshaking, playback/answerback control, message switching and message storage. Message storage is often desirable both because of existing time differences between distant offices and because of transmission speed differences on the international and domestic links.

A typical current implementation of a CU consists of a dedicated rack of special-purpose hardware specifically tailored to a given customer's requirements. The principal sub-assemblies are a message switch with appropriate communication interfaces, code converters, and relatively expensive tape-loop storage media. Also a minimum of two tape loops are required per system since internationally and domestically bound traffic cannot share the same loop. Finally, use of serial storage media necessitates that messages be transmitted in the same order they are received.

The principal undesirable features of hard-wired leased channel implementations are high cost, difficulty in custom-

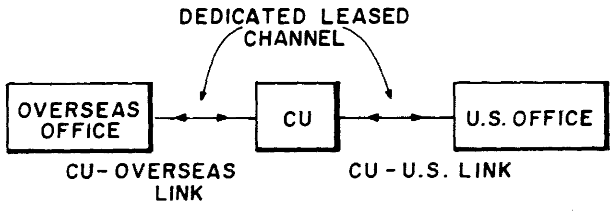


Figure 1—Typical leased channel system

izing a design (e.g., answerback sequences, character expansions, character codes and formats are hard-wired), and high maintenance costs with high mean times to repair. Furthermore, many desirable system functions are prohibitive to implement via random logic.

MICROPROCESSOR IMPLEMENTATION

The microprocessor based leased channel control unit described in this paper is presented in Figure 2. It consists primarily of three basic components: an LSI microprocessor, a floppy disc drive and its associated processor interface, and a pair of communications line interfaces. The communications interfaces support simultaneous full duplux asynchronous data transmission. Modular design minimizes the hardware changes required to support a wide variety of communications channels and devices. Parameters such as data rate, transmission mode, character length, and parity are programmable. The microprocessor, called COSMAC, is an RCA proprietary, byte-oriented, parallel structure machine. Even though COSMAC can directly address 65 Kbytes of random access memory, only 4 Kbytes are used in our current design. The TV display and keyboard are not essential parts of the system but can be used to display status information and to provide the system with operator interaction and diagnostic capability when required. The floppy disc used in the current design is a Century Data Systems CDS-110 and as implemented, has a storage capacity of 118 Kbytes.

The leased channel system depicted in Figure 2 can duplicate all the system functions available in a hard-wired design at much lower cost. Additionally, many new and highly desirable system functions and features are now available at no extra cost in hardware. These are summarized in Table I.

TABLE I—Advantages of Microprocessor Implementation

1. Rapid Implementation of Customer Requirements
2. Dynamic Memory Allocation
3. Message Priority Queueing
4. Programmable System Specifications
   —Data Rates
   —Character Formats and Codes
   —Character Expansion
   —Playback/Answerback Control
   —Error Detection/Correction
5. Maintenance
   —Single Unit Backup
   —Fault Isolation
   —Diagnostics
   —Rapid System Regeneration
6. Data/Message Logging and Archival Storage
7. Improved Cost/Performance

Notice that the functions and features given in Table I are in addition to the functional capabilities of typical current implementations.

Rapid implementation of customer requirements

Since system control resides in a RAM it is extremely flexible because the basic system can be easily tailored to individual customer needs by software modification. Thus the time needed to get a new customer on-line is greatly reduced. Furthermore, system upgrading in the field is made possible by modular software design.

Dynamic memory allocation/message priority queueing

As previously discussed, where message storage is required, two tape loops are currently employed—one for the domestic bound and one for the overseas bound traffic. Traffic flow imbalances between domestic and overseas traffic often result in heavy loading in one direction. This is currently resolved by adding an additional tape loop storage unit to the heavy traffic direction. Meanwhile, the tape loop associated with the lighter traffic remains essentially unused. The use of a random access storage medium (floppy disc) results in the dynamic allocation of memory wherever it is needed—thus if the traffic is heavier in one direction, that direction will be assigned more memory.

An additional desirable feature available for free is that of message priority queuing. Since messages are stored on a random access device, one can tag each message with a priority. Thus when messages are requested from the system, those with highest priority can be transmitted first. Contrast this with the first-in-first-out (FIFO) requirement associated with tape loops. Finally, individual messages can be retransmitted as needed without having to retransmit the entire stream.

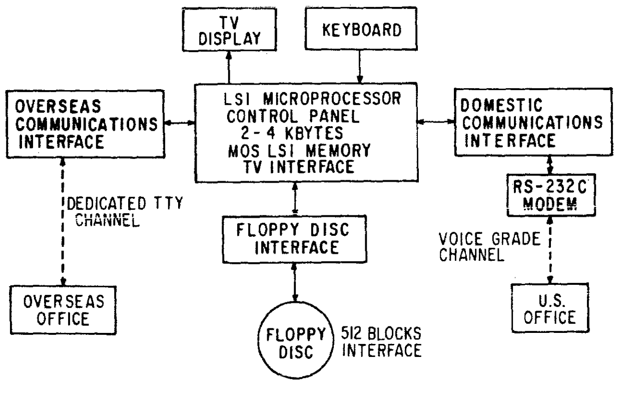


Figure 2—Leased channel control unit

Programmable system specifications

System specifications such as demestic and overseas data rates, character codes and formats, control characters, playback/answerback character sequence generation/detection and character expansion can all be implemented by simple software changes. Character expansion is necessitated by the use of 5-level (baudot) codes in international data communications. Thus many ASCII characters have no baudot counterparts and are represented by sequences of valid characters. For example, a customer may request that $(ASCII) expands into DOLLAR (baudot). Finally, error detection/correction algorithms can be implemented or modified by changes in the software.

Maintenance

Systems tailored to specific customer requirements will differ only in the software. Thus only single unit backup needs to be maintained. When a system component fails (e.g., cpu, disc, etc.), simply plug in a new component, reload the software and go. To facilitate the latter function, the disc interface supports a bootstrap function which permits system regeneration in seconds. This feature will be discussed more fully in a following section on the disc interface organization.

A system structure based around a cpu lends itself readily to self-testing. Diagnostic programs can be run to isolate and identify faulty system modules. A keyboard and TV display permit the operator to interact with the diagnostic programs and rapidly determine which portion of the hardware is inoperative. Note that this diagnostic testing can be done off-line, since the load of the faulty system can be taken up by an identical (hardware wise) backup system.

COSMAC

The COSMAC microprocessor was designed for implementation on a single 40-pin, MOS/LSI chip. It provides a flexible, powerful, building block for a variety of stored program products, including device controllers, terminals, and computers. Special features such as "on chip" DMA channel minimize added circuits for complete systems. A parallel internal structure using static circuits provides maximum reliability, speed, testability, and application flexibility. Proprietary architecture, utilizing a one byte instruction format, minimizes program memory requirements.

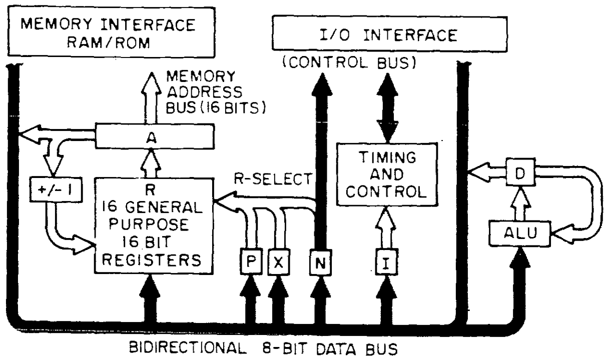

MEMORY INTERFACE RAM/ROM
I/O INTERFACE
(CONTROL BUS)
MEMORY ADDRESS BUS (16 BITS)
A
+/- 1
R 16 GENERAL PURPOSE 16 BIT REGISTERS
R-SELECT
TIMING AND CONTROL
D
ALU
P
X
N
I
BIDIRECTIONAL 8-BIT DATA BUS

Figure 3—COSMAC microcomputer architecture

Figure 3 illustrates the microcomputer architecture. "R" represents an array of sixteen, 16-bit general purpose registers. (This is essentially a 16 × 16 bit RAM.)

P, X, and N are three 4-bit registers. The contents of P X, or N select one of the 16 R registers. R(N) will be used to, denote the specific R register selected by the 4 bit hex digit contained in the N register. RO(N) denotes the low order 8 bits (byte) of the R register selected by N. R1(N) denotes the high order byte. The contents of a selected R register (2 bytes) can be transferred to the A register. The 16 bits in A are used to address an external memory byte via an 8-bit multiplexed memory address bus. The 16-bit word in A can be incremented or decremented by "1" and written back into a selected R register.

M(R(N)) refers to a one byte memory location addressed by the contents of R(N). This indirect addressing system is basic to the simplicity and flexibility of the architecture.

D is an 8 bit register that functions as an accumulator. The ALU is an 8 bit logic network, I is a four bit instruction register. Bytes can be read onto the common data bus from any of the registers, external memory, or input/output devices. A data bus byte can, in turn, be transferred to a register, memory, or input/output device.

The operation of the microcomputer is best described in terms of its instruction set. A one byte instruction format is used as shown in Figure 4. The instructions are summarized below where [XX] contains two hex digits and represents the instruction byte.

Register Operations

[1 N]	Increment R(N) by 1
[2 N]	Decrement R(N) by 1
[8 N]	Transfer RO(N) to D
[9 N]	Transfer R1(N) to D
[A N]	Transfer D to R0(N)
[B N]	Transfer D to R1(N)
[C N]	Transfer D0 to R00(N)

Memory Operations

[4 N]	Load D from M(R(N)) and increment R(N)
[5 N]	Store D in M(R(N))

Miscellaneous Operations

[0 N]	Idle
[3 N]	Branch
[6 N]	Input/output byte transfer
[7 N]	Interrupt control
[D N]	Set P to value in N
[E N]	Set X to value in N
[F N]	ALU operations

For the miscellaneous operations, N no longer selects one

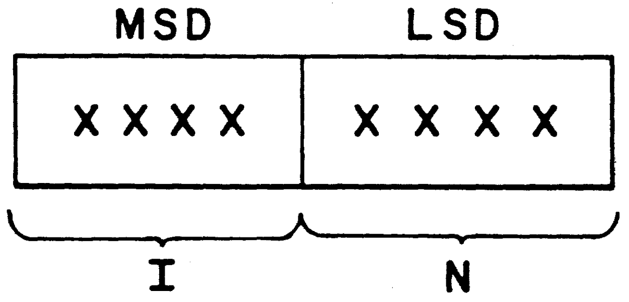


Figure 4—One byte instruction format

of the R registers, but is decoded as needed. For example, for instruction "3", N selects the type of branch instruction desired.

Instruction "6" permits byte transfers between memory and input/output devices via the common byte bus. The value of N specifies the direction of the byte transfer. M(R(X)) can be sent to an input/output device or an input/output byte can be stored at M(R(X)). The digit in N is made available externally during execution of the input/output byte transfer instruction. This digit code can be used by external I/O device logic to interpret the common bus byte. For example, specific N codes might specify that an output byte be interpreted as an I/O device selection code, a control code, or a data byte. Other N codes might cause status or data bytes to be supplied by an I/O device. COSMAC can directly address up to 65K bytes of RAM or ROM, has a program execution speed of up to 100,000 instructions/second and can achieve a DMA burst transfer rate of up to 200,000 bytes/second. Additional and more detailed information on the COSMAC microprocessor architecture and its instruction set is available in References 1 and 2.

Our experience, to date, indicates that COSMAC is indeed very well suited to the types of processing required in data communications systems (table look-up, interrupt driven software, data management). Multiply and divide must be done in software, but these operations are uncommon in low end data communications systems.

COMMUNICATIONS INTERFACES

An interface links the microprocessor to each communications channel. The domestic interface connects to a voice-grade telephone line via an RS-232C compatible modem or data set. The overseas interface connects directly to overseas channel terminal equipment which accepts TTY current-loop signals.

Both interfaces may be active simultaneously supporting full-duplex asynchronous data transmission. They perform serial-to-parallel and parallel-to-serial conversion, parity generation and checking, and character synchronization and buffering. For convenience, status display registers are also included in the communications interfaces. To maintain maximum flexibility, all other communications functions, such as code conversion and control character recognition, are performed in software. More detailed information on the operation and implementation of the communications interfaces is presented in a companion paper.[3]

FLOPPY DISC INTERFACE

Floppy discs

Since the commercial introduction of floppy discs in the latter part of 1972, the number of announced drives has increased from two (CDS and Memorex) to almost a dozen. Use of floppy discs in systems such as the IBM 3740 Data Entry System attests to the floppy disc drive's basic simplicity, low cost (potentially much lower) and applicability to many low end systems.

Properties common to most floppy disc drives include the following: The recording medium is a non-volatile, flexible, small (7.5" dia.) oxide coated disc, usually packeted in an envelope. Data is recorded on only one side of the disc. Typically, a one inch recording band is accessible through an aperture in the envelope. During reading or writing, contact is usually made between the head and medium (in some cases, a "fragile" air bearing exists). Since some contact exists, precise "flying head" and mechanical stability problems are avoided—however, head and medium wear do occur and must be accounted for (usually by maintaining head/disc contact only during reading and writing). The disc rotates at slow speeds ranging from 90 to 400 RPM, can store typically 0.5 to 2.5 Mbits, has an average access time of about 500 milliseconds, can transfer data at 33-250 Kbits/second, and costs about $5. Changing cartridges can be accomplished in seconds. More

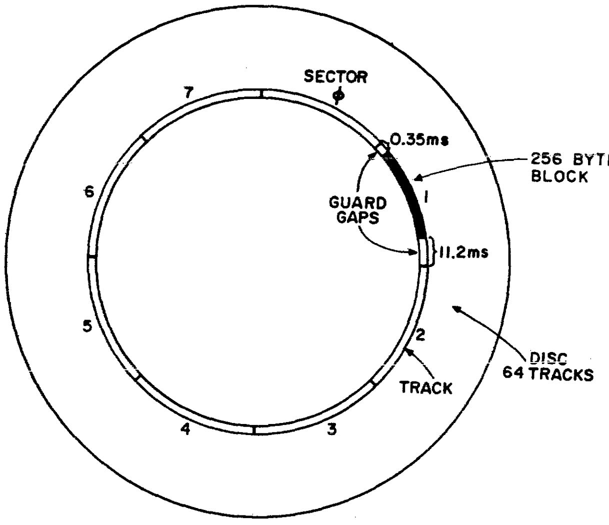


Figure 5—Floppy disc data structure

information on floppy discs is available in References 4 through 6.

Interface organization

Appendix A presents a block diagram of the floppy disc (FD) interface and discusses the hardware implementation. The philosophy behind the interface architecture was motivated by our leased channel project. A substantial portion of the CPU's processing power will be needed to support the communications interfaces. Thus it was decided to dedicate the CPU's Direct Memory Access (DMA) channel to the FD whenever I/O to/from the disc is required. Of course, the DMA is available to other devices (such as the TV display) whenever the FD is not busy. With the above philosophy, the CPU need only issue a few instructions and then check, periodically, to see if the data transfer is completed.

Four instructions need to be issued by the CPU to effect data transfer. These will select the FD, load appropriate status information into a two-byte buffer (2 instructions) and start the I/O operation. When block transmission is completed, a flag is raised which can be tested by the CPU.

Each disc sector (8 sectors/track, 64 tracks/disc) will contain *one* 256 byte block, see Figure 5. Each block consists of a 16 byte synchronization pattern, 232 data bytes and an 8 byte trailing pattern, Figure 6. The total disc capacity is thus 512 blocks or 131,072 bytes (118,784 data bytes). There are 9 bits/byte—8 data bits plus parity. This block per sector approach greatly simplifies the interface electronics and seems to be a useful organization for most low cost random access bulk storage systems envisioned.

The selection of a 256 byte block size in conjunction with the present design allows for an 11 msec guard gap following the data block before the next sector is reached. Thus there is enough time for the software to access the very next sector should data chaining be desired.

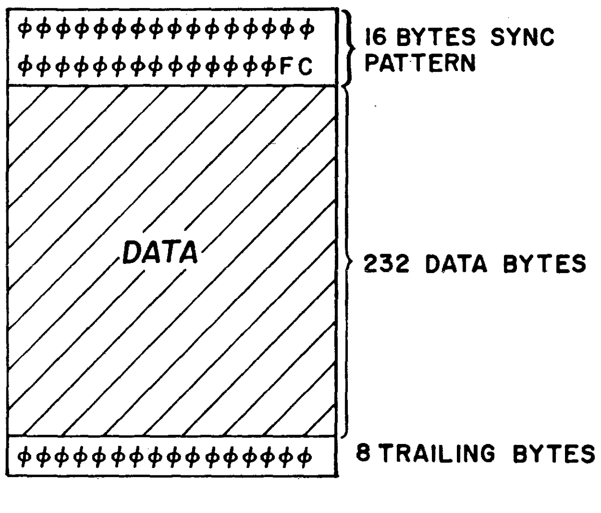


Figure 6—Data block structure

The present interface is equipped with a control panel which, among other things, resets the head over track ØØ. A bootstrap feature enables the user to enter a loader program (resident on the disc) into the CPU's memory at the flick of a switch. This loader can then load the CPU's memory with any other program residing on disc, and renders the system completely self-contained insofar as Initial Program Loading (IPL) is concerned.

Control buffer

Since the FD interface must work independently of the CPU, it must initially be provided with status information. This status information must include the following:

—track information
—sector information
—read/write information

Status information is stored in the interface in a two byte control buffer. The bit assignments are presented in Figure 7.

Use of the control buffer in the FD interface enables it to work completely independently from the CPU since all the information the interface needs is continuously available to it.

Disc related CPU instructions

In our current implementation, a 61 instruction "selects" the peripheral device to or from which information is to be transferred. Device selection is accomplished by assigning each I/O device a "device number" and ensuring that M(R(X)) contains the desired device number when the "61" instruction is executed. The five CPU instructions needed for disc/CPU communication are as follows:

I	M(R(X))	Function
61	08(hex)	Select FD
62	11XXXXXX	Load Buffer A
62	Ø1XXXXXX	Load Buffer B
62	1ØXXXXXX	Start I/O
62	ØØXXXXXX	Turn FD off

When a start I/O instruction is issued, the head moves over the desired track and the correct sector is located. Simultaneously, the head is loaded (contact with disc is made) and all suitable stabilization delays are generated. When the desired sector reaches the head and all stabilization delays have elapsed, the interface will raise the IN REQ or OUT REQ lines of the DMA either wishing to store a byte in M(R(Ø)) or requesting a byte from M(R(Ø)). See References 1 and 2 for details on the operation of the CPU's DMA channel. When one data block has been trans-

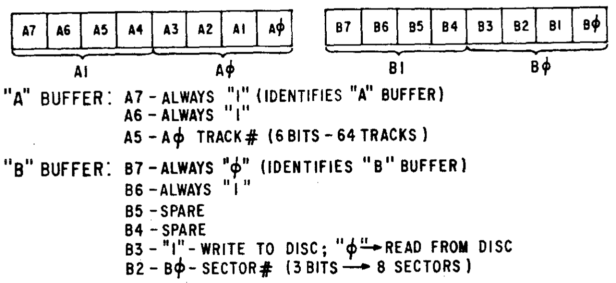

A7 A6 A5 A4 A3 A2 A1 AØ
B7 B6 B5 B4 B3 B2 B1 BØ
A1
AØ
B1
BØ
"A" BUFFER: A7 - ALWAYS "1" (IDENTIFIES "A" BUFFER)
A6 - ALWAYS "1"
A5 - AØ TRACK # (6 BITS - 64 TRACKS)
"B" BUFFER: B7 - ALWAYS "Ø" (IDENTIFIES "B" BUFFER)
B6 - ALWAYS "1"
B5 - SPARE
B4 - SPARE
B3 - "1" - WRITE TO DISC; "Ø" → READ FROM DISC
B2 - BØ - SECTOR # (3 BITS → 8 SECTORS)

Figure 7—Control buffer

mitted to/from the disc, an external flag is raised which can periodically be tested by the CPU. If the block transfer is complete and if the operation was a disc read, the CPU can then test a different external flag to determine whether a parity error has occurred. If the software desires to fetch the block of data stored in the next sector (or any other sector on the same track) the software can issue a 62 instruction with M(R(X)) = Ø1XXXXXX to update control buffer "B" followed by a 62 instruction with M(R(X)) = 1ØXXXXXX to start another block transfer. The 11.2 ms. guard gap following the data block in each sector is sufficiently large to enable the software to catch the data block on the very next sector! If the software no longer needs the disc, a 62 instruction with M(R(X)) = ØØXXXXXX is issued to turn the device off.

I/O transfer rate

The present design results in an average access time (seek + latency) of 560 ms. and a data block transfer time of about 70 ms. for a total average transfer and access time of 630 ms. Raw data transfer between the FD interface and the disc itself occurs at 33 Kbits/sec. Insofar as the CPU is concerned, the critical I/O rate is in *bytes/second* flowing between the CPU ahd interface. Thus the burst I/O rate is given by 33K/9 = 3.7 Kbytes/second. Hence less than 2 percent of CPU cycles are stolen during disc data transfer.

Bootstrap loader

In the following it is assumed that sector Ø of track ØØ contains a bootstrap program which can be used to load programs previously stored on disc into the main memory. The programming convention required for the programs residing on disc is given in Appendix B. The structure of the control buffers is such that when they are cleared (by a CPU reset), they point at sector Ø, track ØØ and are in the "read" mode. The disc interface is organized such that the flick of a switch will select the disc and issue a false "start I/O" instruction which will bring the loader into main memory. Any program residing on disc can then be loaded via this loader program eliminating the need for auxiliary program load devices such as cassettes or paper tape and greatly simplifying system regeneration after a crash.

CONCLUDING REMARKS

The advent of low cost LSI microprocessors and bulk storage devices will bring about a profound change in the architecture of next generation low end data communication systems. Microprocessor based data communications, typified by the leased channel system described in this paper, will begin to proliferate before the end of this decade. Many desirable system functions and features will be made economically viable via the stored program approach. Functions such as programmability, dynamic memory allocation, message priority queuing, etc., have already been discussed. However, system features such as ease of maintenance, which are of prime importance to the operator, rarely receive sufficient attention from system designers. A microprocessor based system, with modular hardware, can recover from a failure in minutes instead of days, as is sometimes necessary in the repair of customized hard-wired logic. Simple substitution can identify the faulty hardware, and the system can be regenerated in seconds.

Looking into the future, many other microprocessor based data communications systems can be envisioned. Intelligent multiplexors, buffers, concentrators, code/speed converters, and/or any combination of the above are but a partial list. These systems are certainly technically feasible and may operate either on a stand-alone basis or as adjuncts to larger computer based data communications systems.

The advent of commercially available all digital communications channels may encourage the development of intelligent repeaters where error detection/correction algorithms, speed/code conversion, buffering, multiplexing, concentration and routing can all be implemented (and readily modified) via software. As new applications and mass markets emerge for low end microprocessor based systems of all types, new LSI CPU's and matching low cost peripheral devices will certainly become available. These in turn will spur the development of multi-microprocessor systems where many of the peripheral device controllers will themselves consist of dedicated CPU's. Thus it seems reasonable that this decade will witness the introduction of complete low-end computer systems, including CPU, display, simple keyboard and mass storage, having manufacturing costs of under $1000. Therein, perhaps, lies the germ for yet another technological mini-revolution that will more than rival the current calculator explosion.

ACKNOWLEDGMENTS

The authors are indebted to R. O. Winder, A, Longo, and J. A. Weisbecker, the inventor of the COSMAC architecture, whose constant support and many hours of discussion have been invaluable in the successful evolution of this system. Finally, the authors wish to thank D. Nichols for his fine work in building the prototype hardware and for his patience and responsiveness in our many design changes.

REFERENCES

1. Weisbecker, J. A., "A Simplified Microcomputer Architecture," *Computer*, February 1974.
2. Swales, N. and J. A. Weisbecker, "COSMAC—A microprocessor for Minimum Cost Systems," *Proceedings of the IEEE International Convention*, New York, N. Y., March 26-29, 1974.
3. Lippman, M. D. and P. M. Russo, "A Microprocessor Controller for International Leased Data Channels," *Proceedings of the IEEE International Conference on Communications*, Minneapolis, Minnesota, June 17-19, 1974.
4. Roessler, G. D., "Selecting a Mass Storage Memory," *EE Systems Engineering Today*, June 1973, pp. 83-86.
5. "Focus on Disc and Drum Memories," *Electronics Design*, Vol. 20, No. 10, May 11, 1972, pp. C16-C22.
6. Davis, Sidney, "Disc Storage for Minicomputer Applications," *Computer Design*, June 1973, pp. 55-66.

APPENDIX A—DISC INTERFACE HARDWARE IMPLEMENTATION

A detailed block diagram of the FD interface logic is presented in Figure 8. Six basic functions can be identified. The control logic is activated either by CPU (normal operation) or by control panel (initial start-up) signals. Except for initial start-up, the control panel is only used to display status information. The control logic acts on information previously stored in the control buffer, simultaneously positioning and loading the head and generating all the required settling delays. When the head is over the desired sector, and all required delays have elapsed, disc reading/writing is initiated. When a disc block read is desired, the sync pattern detector locates the beginning of the data stream and 232 bytes are framed, checked for parity, and transmitted to the CPU via the DMA channel. An external flag is raised to signal end of transmission. When it is desired to write a block to disc, output bytes are immediately requested from the DMA channel. These bytes are, in turn,

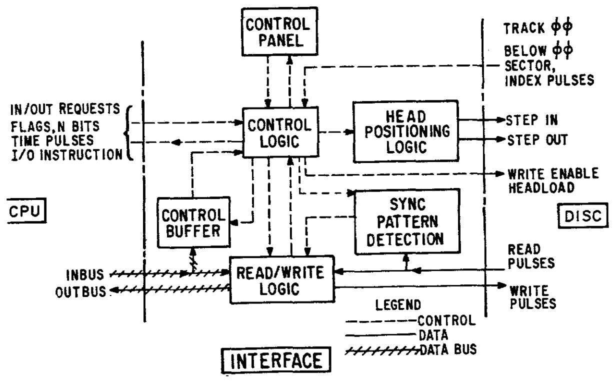


Figure 8—Floppy disc interface organization

serialized and appended with an odd parity bit. When 256 bytes have been written, a flag signaling end of transmission is raised. The end of transmission flag can be tested by the CPU which can then either modify (if needed) the control buffer and issue another start I/O instruction, or issue a "turn FD off" instruction which will, among other things, unload disc head to minimize head/medium wear. Head/disc contact is maintained only when the disc is active.

The current disc interface design can readily handle several disc units by adding simple multiplexing logic.

APPENDIX B—DISC PROGRAMMING CONVENTIONS

Programs must start at memory address hes "ØØ88" (M(ØØ88)). M(ØØØØ) to M(ØØ87) can be used during program execution as temporary storage or as the TV display area. However, program execution must not depend on the initial contents of this area. Utility programs, including the disc loader, will be executed in M(ØØØØ) to M(ØØ87). The last function performed by any utility program is to insert an unconditional branch over the utility code to permit normal program operation.

The multipurpose batch station (MBS) system—Software design

by CLEMENS B. HERGENHAN and MARK M. ROCHKIND

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

INTRODUCTION

In 1971, Bell Laboratories undertook the development of a minicomputer based batch terminal system primarily to support remote computing activities at its many locations. The effort was to result in a high performance, cost effective system which could be used to improve upon an array of specialized terminals leased from a variety of vendors.[2] In addition to functioning as a conventional remote batch terminal, the system, known now as the Multipurpose Batch Station (MBS), was to serve as an interlocation data link, as a local media conversion station and as a generalized communications interface between large central computers and foreign vendor peripherals. In terms of communications support, the MBS System was to accommodate synchronous disciplines, either 2-wire or 4-wire, at bit per second rates from 2,000 to at least 50,000. The system was to be interrupt driven and was to optimize thru-put for voice grade channels.[3,5]

These design objectives were achieved. There are MBS Systems installed today which communicate with both Honeywell and IBM large scale systems; software to support Control Data central systems is under development. The architecture of today's installed set of MBS Systems varies widely. Prospective users select a combination of peripherals determined by their needs. A popular configuration supports a 500 lpm line printer on a communications line. Such a system operates in a self-service polling mode drawing batch output from some central system in support of time-sharing users. More comprehensive configurations support card reader, different types of line printers, card punch, console typewriter (TTY or EIA), paper tape reader/punch, push-buttons, industry compatible 7 and 9 track, 800 bpi magnetic tapes and DECtape for loading library programs. Systems range in cost from $11,000 to more than $60,000, peripherals included.[1] There are more than 20 systems currently installed.[4]

The MBS mainframe functions as a peripheral switch in that it serves principally to interconnect I/O devices. The software design accounts in large measure for the success of the system. Whereas the explicit design of the software was driven by the very primitive character of the mainframe selected, this combination of hardware and software provided a competitive edge which MBS continues to enjoy.

REVIEW OF MAINFRAME ARCHITECTURE

The MBS System is designed around a Digital Equipment Corporation PDP-8E, one of the very early entries in the minicomputer field and still one of the most economical and reliable of small machines. A basic understanding of the operation of the 8E is necessary to appreciate the design of MBS software.[6]

The MBS mainframe operates with 4K, 8K or 12K of 12-bit words depending on the application intended. Each 4K of memory is called a *field*, and each field consists of 32 128-word *pages* (0 thru 31). Memory reference instructions can directly address any location on Page 0 or on the current page of the current memory field. One level of indirection allows a memory reference instruction to access any location in memory.

There are two 12-bit working registers available to the programmer. The AC receives the results of all logical and arithmetic operations. The MQ functions principally in extended arithmetic operations of which no use is made in the MBS design. It is available also as a temporary storage register but because it can communicate only with the AC, it is little used. The L register is a 1-bit extension of the AC used for "carry" purposes.

The instruction counter consists of 2 registers; a total of 15 bits. The 3-bit IF (Instruction Field) register provides the 3 most significant address bits; it selects the field from which the next instruction will be taken. The 12-bit PC register provides the 12 least significant address bits. The 12 least significant bits of the effective address of a memory reference instruction are derived from the instruction word. When an instruction is of the *direct* memory reference type, the IF register provides the 3 most significant bits of the address. For *indirect* memory reference instructions, the 3 most significant bits of the address are provided by the 3-bit DF (Data Field) register. In addition, 8 special memory locations provide an auto-index capability. When addressed *indirectly*, these are automatically incremented before use.

All memory reference instructions indicate the operation code in bits 0-2, a specification of *direct* reference or *indirect* reference in bit 3, a specification of Page 0 or current page in bit 4, and the 7 least significant bits of the address in bits 5-11. There are six memory reference instructions. These

Code	Symbol	Function
0	AND	AC .AND. ⟨ADDR⟩ → AC
1	TAD	AC + ⟨ADDR⟩ → AC
2	ISZ	1 + ⟨ADDR⟩ → ⟨ADDR⟩
		If ⟨ADDR⟩ = 0; PC + 1 → PC
3	DCA	AC → ⟨ADDR⟩; zero → AC
4	JMS	PC → ⟨ADDR⟩; ADDR + 1 → PC
5	JMP	ADDR → PC

Figure 1—The PDP-8 provides 6 memory reference instructions. ADDR = effective address computed at execution time; ⟨ADDR⟩ = contents of effective address

are reviewed in Figure 1 in terms of an effective address ADDR which is calculated at execution time.

Of the two remaining operation codes, #6 (IOT) is used to control I/O devices. Bits 3-11 are used to select the proper device and to initiate functions. The final operation code, #7, is for microinstruction operations involving the AC, L and MQ. Bits 3-11 indicate the operation to be performed, such as load MQ from AC, clear L and AC, increment AC, rotate AC 1 or 2 bits to the left or right, halt, no operation, etc. These bits may also specify that the next instruction should be skipped if certain conditions exist in the AC or L. For example, one can skip on $AC=0$, $L\neq 0$, $AC>0$, $AC<0$ or some combination of these.

The PDP-8E has a single level of interrupt. An interrupt by any device causes a Subroutine Jump (JMS) to location 0 of Field 0. All future interrupts are inhibited until enabled by the program with the Interrupts On (ION) instruction. The program must poll each device to discover which caused the interrupt and to request that device to remove the interrupt. This is typically done with a "skip on flag" instruction (operation code 6) for each device followed directly by a JMP and then by a "clear flag" instruction (also operation code 6).

SOFTWARE DESIGN CONSIDERATIONS

The ability to cope with exception conditions was designed into MBS software. Among the mechanisms used are alarm clock tables which can initiate recovery routines. Robustness has been established by the many field installed systems. The first such system, a self-service card lister, was installed at Murry Hill in 1971. It runs, entirely interrupt driven, for months at a time without requiring a software reload. Another installation which interconnects an IBM 1403 printer to a Honeywell Series 6000 computer is without a card reader. The PDP-8E in use here is served by core memory and the software remains intact even across interruptions in power.

For the PDP-8E, whole word operations are preferable to bit operations. The lack of an OR instruction and index registers make table searching and bit setting difficult. Accessing tables, searching them, and setting and resetting bits take large amounts of code. To make the operating system efficient, we relied on executable tables. Planting JMS instructions in the right places results in simple whole word operations and eliminates scanning tables.

The MBS software had to remain flexible since we expected large growth in the number of peripherals MBS would be asked to accommodate. Today the number has grown to more than a dozen and several communications protocols are supported. The System is interrupt driven with software implemented priorities. This prioritizing of a single level of interrupt follows a scheme used by Honeywell in its Data-Net/355 General Remote Terminal System. The design outlined here using executable tables is especially suitable for very simple computers.

INTERRUPT HANDLING AND INTERRUPT PROCESSING

MBS makes a clear distinction between interrupt handling and interrupt processing. Interrupt handling is defined to be the querying and acknowledging of interrupts on the interrupt bus. Interrupt processing is the response to a specific interrupt for a specific device. The MBS Interrupt Handler acknowledges interrupts from all devices and activates Interrupt Processors in the Master Dispatcher Table. An activated (enabled) Interrupt Processor is one which is a candidate for execution.

Every device on MBS is capable of generating at least one type of interrupt; a request for service. Some devices have several interrupt flags and will generate interrupts to report error conditions. The interrupt handling code begins at location 1 of Field 0.

Before the Interrupt Handler begins the task of scanning the I/O devices, it must save the registers being used by the interrupted code. The Handler saves the PC, MQ, AC and flags (L, IF and DF) in 4 words specified by the Master Dispatcher Table for use later. After all devices are scanned, control is given to the Master Dispatcher Table which will dispatch to the highest priority Interrupt Processor which has been enabled.

The interrupt handling code runs with interrupts inhibited and so is kept to a minimum. Interrupt handling code for most devices consists of resetting the interrupt flag for that device and making an entry in the Master Dispatcher Table (MDT) so that control will pass to an Interrupt Processor to service that specific device.

For a few devices the amount of code to make an entry in the MDT is greater than the code needed to process the interrupt. For such cases interrupt processing occurs during interrupt handling. For example, a clock in MBS used as an interval timer interrupts every 20 milliseconds and requires only that its flag be reset and a word in core be incremented. This is done by the Interrupt Handler.

THE MASTER DISPATCHER TABLE

The MDT is executed rather than searched. Each Interrupt Processor has a 7-word entry reserved for it as shown in Figure 2.

When an Interrupt Processor is neither enabled nor in execution, word 1 is a NOP (No Operation). Execution of word 1 will allow control to pass to word 2 and cause a JMP to the next Interrupt Processor's 7-word entry. When the Interrupt Handler wishes to activate an Interrupt Processor, it overlays the NOP with a JMS DISPAT.

DISPAT is a subroutine which uses the address in its return linkage to access words 3 through 6 of the block which transferred control to it. DISPAT simply restores the MQ, AC and flags, enables interrupts and jumps to where word 3 (the Program Counter save area) is pointing.

Word 3 is set initially to the value of word 7, the Interrupt Processor's entry address. Thus the first time any Interrupt Processor is enabled it will gain control at its entry point. Should an interrupt occur while an Interrupt Processor is in execution, the Interrupt Handler gains control and uses the value in DISPAT to find the proper set of words (3 through 6) in which to do its saving. When this block is encountered and executed again, DISPAT will restore control to the Interrupt Processor where it was last interrupted. Note that DISPAT has no idea whether it is giving control to an Interrupt Processor for the first time (at its entry point) or redispatching at an interrupted stage.

All Interrupt Processors return to a routine called RMDT (Return—Master Dispatcher Table) when they are done. RMDT uses the value in DISPAT to discover which Interrupt Processor has completed its function. It uses this value to locate word 7 of that block and to place it in word 3. This assures that the Interrupt Processor will gain control at its entry point when it is dispatched to next. RMDT changes word 1 back to a NOP and then executes the MDT to give control to another Interrupt Processor which has been enabled. The highest priority Interrupt Processor enabled is always dispatched to.

When no Interrupt Processor requests service, control passes to the Task Dispatcher. This is accomplished by retaining a JMS DISPAT in word 1 of the last MDT entry. The Task Dispatcher never returns to RMDT; it is always enabled—that is, word 1 never becomes a NOP. This unconditional exit from the Master Dispatcher Table means we need no special code to check for the end of table.

The Master Dispatcher consists of 3 parts: MDT—the executable dispatcher table, DISPAT—the subroutine which gives control to an enabled routine, and RMDT—the entry point back into the Master Dispatcher Table.

The dispatching process is quick. DISPAT consists of only 13 instructions and scanning MDT requires only 2 instructions (2.4 microseconds) for each inactivated Interrupt Processor.

Master Dispatcher Table / Entry

1	NOP or JMS DISPAT
2	JMP to next MDT Entry
3	Program Counter save area
4	AC save area
5	MQ save area
6	FLAGS save area
7	Entry Point to MDI Entry

Figure 2—The Master Dispatcher Table consists of 7 word entries as shown. An entry appears for each Interrupt Processor. The earlier in the Table an entry appears, the higher the priority assigned to that entry. The last entry in the Master Dispatcher Table is the Task Dispatcher for which word 1 is always JMS DISPAT

Task Dispatcher Table

```
TDT → NOP or JMS RTDT / for Task #1
      NOP or JMS RTDT / for Task #2
      ⋮
      NOP or JMS RTDT / for Task #n
      JMP TDT
      Pointer to Task #1 Entry Point
      Pointer to Task #2 Entry Point
      ⋮
      Pointer to Task #n Entry Point
```

Figure 3—The Task Dispatcher Table consists of 2n+1 entries where n is the number of tasks to be dispatched to. When the System is quiescent, the processor cycles through the Task Dispatcher Table (TDT) executing NOP's and the JMP TDT. A task is queued by replacing the corresponding NOP with a JMS RTDT

The sequence of entries in the MDT assigns implicit priorities to the Interrupt Processors. If an interrupt occurs while we are executing a low-level Interrupt Processor (one deep in the MDT), we will service a higher level Interrupt Processor before returning to the lower one. It is possible then to have all Interrupt Processors in an interrupted state. By its position in the MDT, the Task Dispatcher ranks as lowest priority; *we only dispatch to tasks when no Interrupt Processors are enabled.*

TASK DISPATCHING

The Task Dispatcher is also an executable table. Each task has 2 words associated with it. The task queue is arranged as shown in Figure 3.

Initially the Task Dispatcher Table (TDT) is all NOP's. Any Interrupt Processor or any task can queue any task by placing a JMS RTDT (Return—Task Dispatcher Table) in the appropriate place in the TDT for that task. The subroutine RTDT knows how many events there are in the system. It adds this to the value in RTDT to obtain the word which will give it the address (and field) of the task being requested. It also stores a NOP on top of the JMS RTDT just executed. Thus, *a task may queue itself.* This also simplifies the return to the Task Dispatcher Table when a task is complete. Upon completion a task executes a JMP to RTDT *indirect*; control will go to the next entry in the TDT to continue scanning for another task. All tasks are of equal priority. Dispatching is done on a round-robin basis.

Since the Task Dispatcher and all subordinate tasks appear to the Master Dispatcher as one large routine (which has lowest priority but is always enabled), an interrupt during any task will cause the Interrupt Handler to save the place in that task in the Task Dispatcher's 7-word entry in the Master Dispatcher Table (see Figure 2). On redispatching to this entry, we return to the interrupted task. Thus no matter which other tasks get queued, *a task always completes execution before another task receives control.* This makes it possible for tasks to share scratch space—a vital property of the design.

When a task queues itself it places itself at the bottom of the task queue. Since the JMP to RTDT *indirect* which is executed to return to the Task Dispatcher Table will bring control into the task queue just one location below the TDT word for the completed task, tasks may queue themselves without fear of lock up—another important feature of the design.

Both the Task Dispatcher Table and the RTDT pointer reside or Page 0 or Field 0 so they are accessible to allmodules.

The RTDT routine (19 instructions) together with the Task Dispatcher Table comprise the Task Dispatcher—which operates with interrupts enabled. It should be clear that when MBS is quiescent, it will cycle through the Task Dispatcher Table executing NOP's and the final JMP TDT.

The pointers to tasks are 1-word entries. A 12-bit address suffices since all tasks are forced to start on even locations. Bit 11 indicates the field and bits 0-10 (with an implied 0 as the last bit) indicate the starting address in that field.

BUFFERS AND MAILBOXES

The strategy for handling buffers derives from an early application of the MBS which required only 4K of core. This application, which interconnected a card reader, a line printer and a panel of pushbuttons, was expanded to include a card punch, a magnetic tape transport and a console typewriter, still within the 4K core limit. When support for synchronous communications was added, a memory extension to 8K was required to accommodate both the additional software and the large buffers needed for efficient data transmission. This separation has been preserved because it allows an MBS without communications to accommodate a wide range of peripheral devices on a 4K mainframe.

There are two pools of buffers. The buffers in Field 1 serve communications and can each be several pages in length. The buffers in Field 0 serve all I/O devices other than communications. To develop a chain of available buffers, the initialization code of MBS divides the unused portion of Field 0 into a linked list of 92-word entries. The 92-word entry will accommodate a 132 character print line plus 3 words of control information.

Tasks employ mailboxes in order to pass buffers between them. A mailbox is a memory cell in which a task expects to find a pointer to a buffer or chain of buffers. Upon completion of some process, tasks will determine whether to requeue themselves by inspecting their mailbox. A task which finds its mailbox empty simply relinquishes control. Any task or Interrupt Processor which fills a mailbox or adds to a buffer chain must queue a task.

ILLUSTRATION OF AN INTERRUPT DRIVEN PROCESS

We consider here how MBS handles one of the simpler peripheral devices: the medium speed (600 lpm) drum printer.

In the MBS System, the drum printer can receive only 3 commands: (1) enter "print" mode, (2) enter "slew" mode, and (3) leave all modes. When directed to enter either mode, the printer responds with an interrupt when ready to receive data—the print line (if it has entered "print" mode) or the slew control (if it has entered "slew" mode). When directed to leave all modes, the function of the current mode is executed, that is, the actual printing or slewing occurs. Data are fed to the printer through the AC by the mainframe processor. The printer can accept characters as fast as they can be loaded into the AC and presented (2.4 microseconds per character).

The task GTPRT is responsible for initiating I/O using the printer. It has a mailbox named PRBUF in which it expects to find a pointer to a chain of buffers to be printed. Assume GTPRT has been queued and a pointer to a buffer is in PRBUF. When GTPRT gains control from RTDT, it removes the top buffer from the PRBUF chain and leaves the pointer in PBUF (a cell which the Interrupt Processor PRINT knows about). GTPRT directs the printer to enter "print" mode and relinquishes control.

Eventually, the printer supplies an interrupt to notify the mainframe that it is ready to accept print line characters. The Interrupt Processor PRINT is activated by the Interrupt Handler and eventually gains control from DISPAT. PRINT feeds data from the buffer pointed to by PBUF, directs the printer to leave all modes, then directs it to enter "slew" mode and returns to RMDT. The line will now be printed. Upon completion of the printing, another interrupt will be generated to indicate that the printer is prepared to receive the slew command. PRINT gains control again and this time causes slew information to be transferred. It also returns the buffer in PBUF to the free buffer chain and clears PBUF. PRINT then queues GTPRT just in case there are more buffers waiting on the PRBUF chain; it directs the printer to leave all modes (so slewing will begin); and it returns to RMDT.

GTPRT will gain control again to initiate I/O for the next buffer in PRBUF if there is one. Any task which places a buffer in the PRBUF mailbox should queue GTPRT just in case the mailbox is currently empty and GTPRT is not in the TDT. Note that queueing a queued event results in overwriting the JMS RTDT in the TDT with another JMS RTDT; it does no harm.

GTPRT may gain control many times between the time it first directs the printer to enter "print" mode and the time PRINT releases the buffer in PBUF. GTPRT always checks PBUF to see if printer I/O is already in progress. If PBUF is non-zero, it relinquishes control to the Task Dispatcher knowing PRINT will queue it when the current printing is complete.

SUMMARY

The Multipurpose Batch Station represents a reduction to practice of a minicomputer system which offers very high performance at very low cost. We have demonstrated that well designed software can compensate more than adequately for primitive hardware—and cost effectively where there are several applications for the developed system. The packing of 1.5 characters per 12-bit word presents no problem and the searching of tables is circumvented by the technique of executing tables outlined in the text. MBS communications software is independent of bit per second rate and can accommodate data transmission speeds in excess of 50,000 bps. The asynchronous handling of peripheral interrupts permits several I/O streams to be serviced simultaneously.

ACKNOWLEDGMENT

The line printer interface described in the text was designed by Peter E. Rosenfeld of Bell Laboratories. Rosenfeld designed other MBS peripheral interfaces as well.

BIBLIOGRAPHY

1. Bowknight, W. J., G. R. Grossman, and P. M. Grothe, "The ARPA Network Terminal System—A New Approach to Network Access," *Proc. Third Data Communications Symposium 73* (1973).
2. Harrison, T. J. and T. J. Pierce, "System Integrity in Small Real-Time Computer Systems," *AFIPS*, 42, 539 (1973).
3. Mills, D. L., "Communications Software," *Proc. IEEE*, 60, 1333 (1972).
4. Mollenauer, J. F., E. J. Sitar, V. B. Turner, "The MINICOM Data Entry System," *AFIPS Proceedings*, Vol. 4B.
5. Newport, C. B. and J. Ryzlak, "Communications Processors," *Proc. IEEE*, 60, 1321 (1972).
6. *Small Computer Handbook*, 1973, Digital Equipment Corporation, Maynard, Massachusetts.

The MINICOM data entry system

by J. F. MOLLENAUER, E. J. SITAR and V. B. TURNER

Bell Laboratories
Murray Hill, New Jersey

INTRODUCTION

The MINICOM system has evolved from the Bell Laboratories Multipurpose Batch Station (MBS) system described in an earlier paper.[1] It is designed to facilitate the entry of data into central site computer systems from laboratory minicomputers. The effort illustrates the growth possible within the MBS framework and makes a new service available to the computer center user.

Central computers as providers of remote services via communications have been known for over a decade. Initially it was assumed that data from laboratory apparatus could be collected directly by large central computers to be processed and stored at some central site. However, this original idea has been considerably modified in recent years.

The principal reason for this change in attitude has been the availability of inexpensive local data processing in the laboratory in the form of the minicomputer. Minicomputers satisfy real time demands but do not provide all the processing power that a user needs. They do provide decentralization of decision making among users, as well as decentralization of disasters. The acceptance of minicomputers has been extensive; various universities, manufacturing installations and research and development laboratories throughout the world have acquired them in quantities ranging into the dozens and hundreds.

Given that data lines to large central computer systems have been available for many years, the number of minicomputers that have taken advantage of them has been remarkably small. The potential savings in data storage and program development costs have likely been outweighed by the difficulty of programming the minicomputer for data communications and by the spotty records of computing centers in providing continuously available service. Those systems that have been tied in have generally been the work of computer enthusiasts more challenged than put off by the communication protocols set up by manufacturers of large systems.

With the costs of logic and storage heading consistently down and salaries up, we expect to see both the number of minicomputer systems and the costs of operating each of them increasing. In fact, we have seen not only the proliferation of minicomputers but also a gradually increasing cost per system as more and more peripherals are being added. It has become increasingly desirable to provide centralized support to the minicomputer community. MINICOM provides this support in the form of data communications between minicomputers and a large central system, reducing the necessity for secondary storage and expensive and hard to maintain peripherals on the laboratory systems.

DESIGN GOALS FOR MINICOM

In designing the MINICOM system, four primary goals were adopted:

1. High reliability
2. Ease of use
3. Low cost
4. Adaptability

High reliability is vital for the minicomputer system which must dispose of its collected data in order to collect more. Central computer system downtime can be disastrous. MINICOM uses a PDP-8E with a disc file to collect data arriving via communication lines. When a user's transmission is complete, his data are queued for transmission at 9600 bps to the central site computer. Should the central computer be unable to accept the data, they remain on the MINICOM disc until the central site becomes available.

Ease of use has been attacked, in a departure from common computer center practice, by providing software for the user's minicomputer, enabling it to use the MINICOM system with little or no programming effort. Subroutines callable by the user's programs will be provided for the more popular minicomputers. These subroutines will drive synchronous communication line interfaces either provided by the computer manufacturer or locally designed.

Synchronous communication is preferred to asynchronous communication because it is more efficient and is relatively open-ended in bit per second rate. The objective of low cost is achieved through the use of voice grade facilities, either dial-up or private line. The investment in data communications hardware can range from low priced 2000 bps devices to more elaborate units operating at tens of thousands of bits per second. The route taken by data going back to the

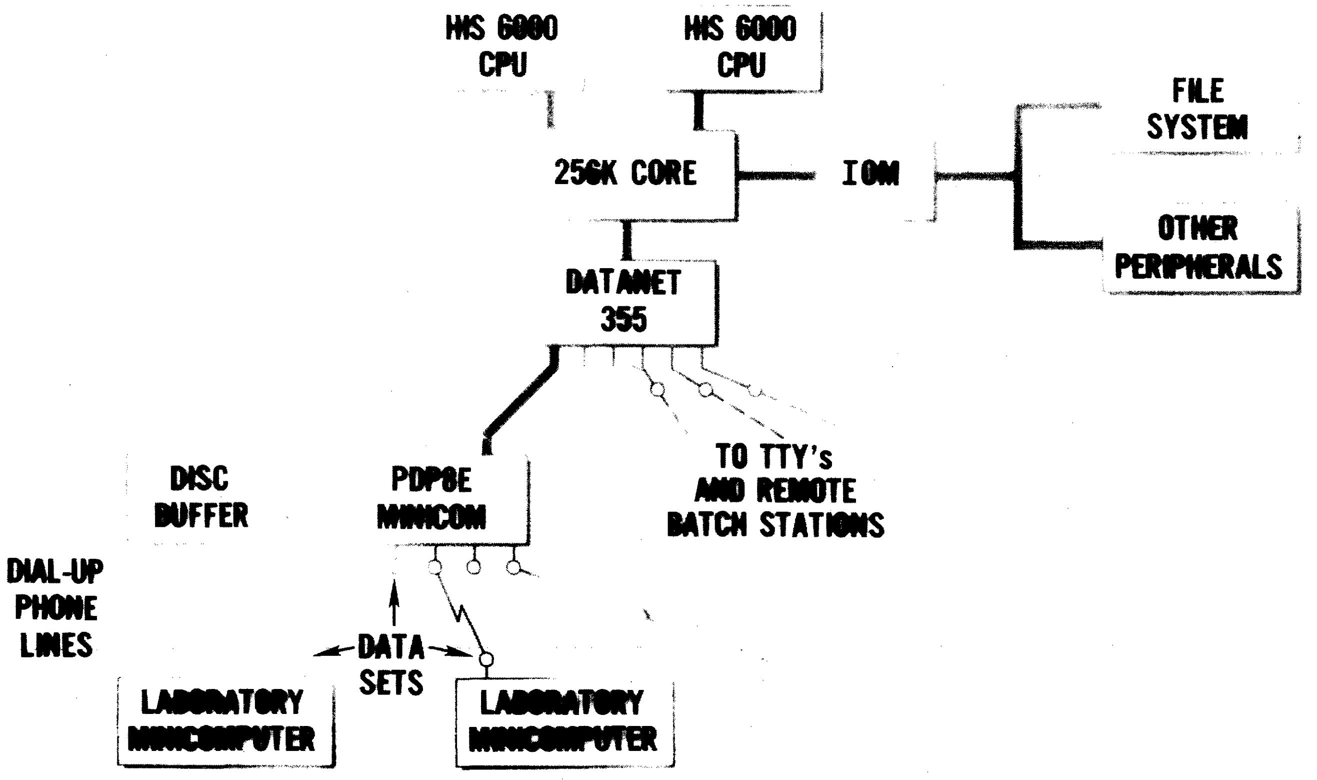


Figure 1—Data path from minicomputer to central system via MINICOM. Not all components of the H6000 are shown

user does not include a stopover on the disc; if the central system is not available, a user waiting to draw data must try again.

Adaptability built into the central design of MINICOM will permit the system to be used as a front end for several large systems. The initial implementation is being done for Honeywell Series 6000 equipment. An IBM implementation will follow, with the same user interface.

SYSTEM OVERVIEW

The MINICOM system enables the user to make several different requests to the central site computer as indicated by request codes in the message header:

1. Append data to a file
2. Erase file and write data
3. Start a batch job to process data sent
4. Draw output from the central computer

Even though MINICOM will support as many as 6 synchronous lines concurrently, we are unwilling to dedicate a line to a user for a long period of time. The intended mode of use is not interactive; the user machine will be connected to send or receive data and will then be disconnected in order to free the line.

The overall MINICOM system is indicated schematically in Figure 1. A group of five lines is available for users calling in to the MINICOM computer, a PDP-8E with 12K of 12-bit core. A sixth communication line is connected directly to the Datanet 355 of the H6000 complex. This connection mates the normal EIA modem interfaces of the two computers by exchanging the data in and out signals and appropriate control leads. The Datanet supplies clock pulses.

The disc system uses two removable cartridges, each with a capacity of 1.6 million words. The use of two drives makes it possible to replace full cartridges with empty ones and makes the capacity of the system virtually unlimited in the event of extensive downtime on the central system. A teletypewriter used as a console completes the hardware configuration.

THE MINICOM SOFTWARE STRUCTURE

The object in using a PDP-8 for MINICOM, rather than any of the newer 16-bit machines, was to take advantage of existing code and the familiarity that had been gained with this machine in the course of earlier MBS work, as well as to use the available central site support, including cross-assembler, macros, library, and linking loader. In terms of software, taking advantage of the MBS system included preserving its nucleus of interrupt and task processing routines and adding a file system and disc handlers within a multiprogramming framework.

The change in emphasis was rather profound: from single stream processing of data in MBS (see Figure 2) to a multiprogramming environment including secondary storage. In MBS, data proceed from a source such as a card reader to the destination on the communication line. When a card is read, for example, an appropriate task is queued. When this task finishes, it may link a buffer of processed data (circles in Figure 2) to another task and place that task in the round-robin task queue. All tasks proceed to completion (except for time out to service interrupts) without waiting for an I/O action to complete.

In the case of MINICOM, several users may send in data simultaneously; the processing invoked by the receipt of a block of data on the communication line will involve disc I/O. Because of the presence of several simultaneous users, tasks have to be re-entrant at any point that disc I/O is requested. It is no longer possible for a given routine to proceed to completion without waiting for I/O. While in principle it would have been possible to write every leg of the flow chart between instances of disc I/O as a separate task, the number of tasks would have been intractably high. The code would have been difficult to understand, debug, and maintain. As a result, two special routines were developed, permitting tasks to make requests to another level of the system for disc I/O. One of these routines is a disc subroutine callable from any task; the other is a special task to cause re-entry to the calling task after its I/O is complete.

The common disc I/O subroutine marks the requesting user dismissed for I/O and issues the disc commands if the disc is not busy. If the disc command can be issued, the user's status is set to "I/O in progress," otherwise to "waiting for I/O." The disc subroutine does not return to the calling task but exists directly to the task dispatcher, which initiates the next waiting task in round-robin sequence. No further

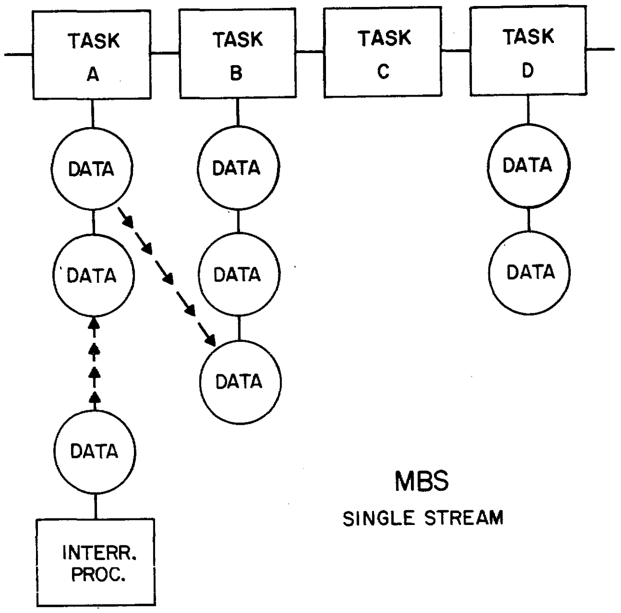


Figure 2—Data flow through original MBS system

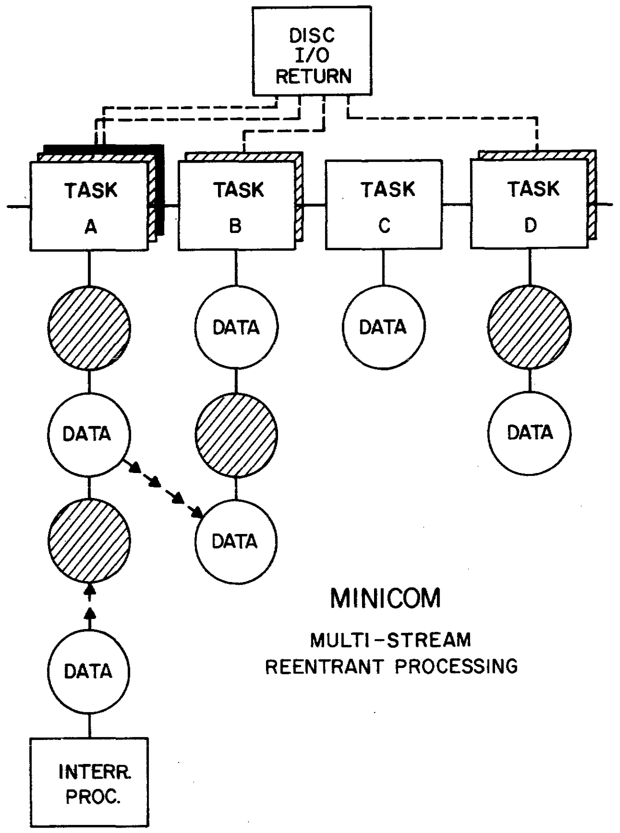


Figure 3—Data flow through MINICOM; data buffers belonging to users dismissed for disc I/O are shaded

file system work is initiated for the user, although he may continue to send data in by communication line.

When a disc I/O request is complete, an interrupt occurs and the disc interrupt processor is entered. (Since it has the highest priority, it preempts control from any other interrupt processor or task.) If any disc I/O is waiting to start, commands are issued for the first requesting user in circular order after the one whose I/O operation is completing. The latter user's status is changed to "I/O in progress," and the requestor of the first I/O is set to "I/O completed." The interrupt processor then queues the disc return task and control reverts to the routine (a task or a lower-priority interrupt processor) that was executing when the disc interrupt occurred.

The function of the disc return task is to capture control at the task level, after all interrupt processors and other tasks have completed normally. On entry, it searches for a user with disc I/O complete, again preserving equal priority among users by going in circular order, and updates his status to "return from I/O." From his disc status table it picks up the number of the task in which the disc I/O was requested and jumps back into the original task at the instruction following the call for I/O.

The foregoing sequence is not markedly different in function from that employed on any multiprogramming system

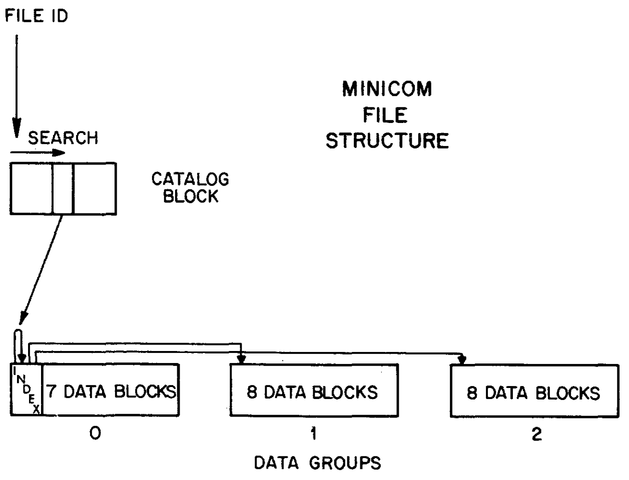


Figure 4—Minicom file system structure; a file with three data groups is shown

with a disc. It should be noted, though, that it conforms to the original interrupt/task processing structure of the MBS systems despite the changed requirements. Figure 3 indicates the multiple-stream processing schematically, for contrast with Figure 2. Reentrant use of the tasks is indicated by multiple overlays of the boxes, although only one copy of the code remains in core. Separate tables are maintained for each user for file and I/O status and for subroutine return stacking. Data queued for the various tasks may be inactive (shaded) because the user owning the data is dismissed for disc I/O. The disc I/O return routine is shown out of the main stream because of its special function of returning control to the task which requested the I/O.

The same mechanism enables a task to yield control when it is denied write access to a table that is being copied to disc. The disc return task is queued for it and control passes back to the task dispatcher. The process is repeated until the table is available.

FILE SYSTEM

In contrast to most file systems, the MINICOM file system was designed for data that are to be read only once. In this case, efficiency of writing is as important as efficiency of reading. In addition, the fact that the disc cartridges might be removed has required that each cartridge be totally self-contained, with its own directory, indices, and available-space map.

The file system is illustrated in Figure 4. Each file is given an identification number in sequence, and the least significant five bits are used to get the master catalog block number in which it resides on the currently active cartridge. This scheme is used in preference to any more elaborate hashing because it permits the same master catalog block to remain in core for use by a number of successive files. The master catalog contains a pointer to the index, which is the first sector of the file. In order to minimize the time required to write on a file, space is allocated in groups of eight 256-word sectors. Each time space is allocated or released, the available space table is recopied to the disc. The index sector contains pointers to the data groups, each of which except the first contains eight data sectors. End-of-file is not marked as such but is indicated in the sector count written in the master catalog entry when the file is closed.

When the final sector of a file is written, it is marked ready for transmission and the transmission task is queued. This routine checks the master catalog for completed files; when it finds one it marks it open to read, then reads and transmits it to the central system sector by sector. When it has successfully transmitted the file, it queues it for release. Files are sent to the central system one at a time in order to make recovery from a central system (or MINICOM) crash easier and more certain. Forwarding buffers in the order received would have permitted a much simpler file system but at excessive cost in reliability.

EFFECTS OF MINICOM USAGE

Anticipated usage of the central computer facility through MINICOM falls into three categories: as a destination for data collected by minicomputers, as a source of input data, and as a source of programs. Since the flow of data from the central machine has no disc buffering, downtime will cause inconvenience to minicomputer users of this mode just as it does to time-sharing users. For inputting data, MINICOM is expected to provide long uninterrupted periods of service owing to the dedicated nature of the application.

We are convinced that minicomputers will require greater levels of support than computer centers have given them in the past. MINICOM provides only one component of this support. In the future, it is clear that minicomputer support will constitute an important part of the computer center's mission.

ACKNOWLEDGMENTS

This work depends on many contributions from the MBS effort described in the previous paper, as well as on many additional contributions by C. B. Hergenhan in the areas of communication line service and H6000 support.

REFERENCE

1. Rochkind, M. M. and C. B. Hergenhan, "The Multipurpose Batch Station (MBS) System: Software Design," these proceedings.

The implementation of the PROPHET system*

by P. A. CASTLEMAN, C. H. RUSSELL, F. N. WEBB, C. A. HOLLISTER, J. R. SIEGEL, S. R. ZDONIK and D. M. FRAM

Bolt Beranek and Newman, Inc.
Cambridge, Massachusetts

INTRODUCTION

PROPHET is a computer system designed to augment the information-handling capabilities of individual scientists. It is oriented toward the needs of pharmacologists, medicinal chemists, and other scientists who study how chemical substances affect biological activity. The system hardware consists of a large time-sharing computer connected over telephone lines to display terminals (Figure 1) in medical schools, hospitals, and other pharmacological research centers. PROPHET's software provides a computing environment, tailored to deal with chemical and biological information, which allows non-programming users to perform complex operations and also supports programming users who wish to extend the system in new ways. PROPHET's purpose is to aid individual research rather than to distribute existing data bases; it does, however, possess capabilities and data structures to facilitate the sharing of data and algorithms between users.

The system is currently in service operation with approximately thirty users at five sites (Figure 2). Typical uses of the system include organizing and filing experimental results; analyzing data through the system's graphing and statistics commands and through user-written procedures; and manipulating models of molecular structures. Specific biomedical research interests of the individual users are outlined in Figure 3.

The impetus for PROPHET development has come from Dr. William Raub of the Division of Research Resources of the National Institutes of Health. PROPHET is being designed and built by the Medical Computer Systems group of Bolt Beranek and Newman Inc. in collaboration with the NIH. The greater part of the system design[1] was performed in 1968 and 1969; prototypes of the basic capabilities were implemented in 1970, and an initial service version was completed in early 1972. Current efforts are directed toward maintaining and refining the system in a demanding user environment and toward adding new applications. The operation and maintenance of the computer equipment and the time-sharing monitor are performed by a separate organization, First Data Corporation.

This paper describes the major implementation approaches selected in the construction of PROPHET. It focuses on the ways in which a variety of computer-science techniques (including time-sharing, extensible languages, interactive graphics, data management, graph-matching, molecular model building, and statistics) were adapted and integrated into a single system to aid drug research. We hope that this discussion of the capabilities and limitations resulting from our implementation choices will be helpful in designing related systems. For example, we believe that this is the first large *service* system to be based on an extensible language, and that our experiences will be useful to others considering the technique.

This paper is organized into two parts: The first part gives a general overview of the PROPHET System, including an example of its use; the second part is a technical discussion of the implementation of PROPHET.

AN OVERVIEW OF PROPHET

A major premise of the PROPHET design is that pharmacologists employing computer aids should be able to work directly with entities related to their research. PROPHET is structured around a set of applications-oriented *data types*.[3] This set currently includes "molecule," "data table," "graph," and "statistical sample," in addition to such types as fixed and floating point numbers, text, Boolean variables, bit strings, and procedures. The existence of a data type implies the creation of a computer representation of that entity (e.g., a molecule as a set of atom and bond representations, which are in turn broken down into types, positions, connectivity, and so forth). It is not intended that either the set of available data types or their representations be fixed; they are designed to grow according to the research interests of the system users. Accommodating this growth gracefully is a major implementation goal. The existence of a data type also implies capabilities for making up instances of that type from user input or from existing entities, for displaying and filing the type, and for performing useful

* The work described in this paper was supported by the National Institutes of Health, under contract number NIH-DRR-70-4113.

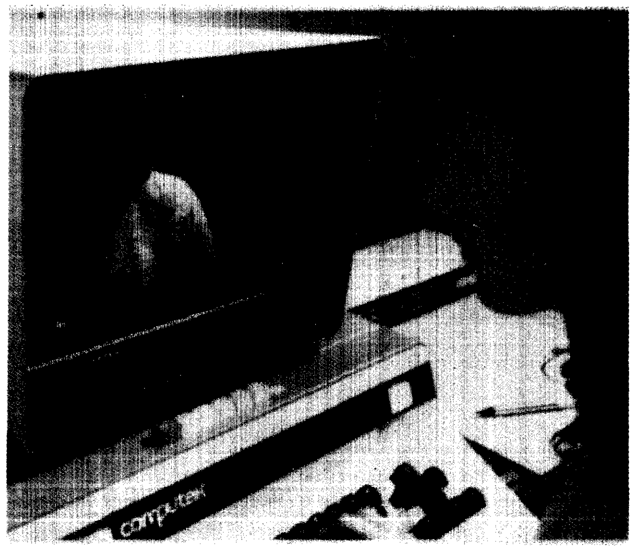


Figure 1—The PROPHET terminal consists of a storage-tube display unit for typing commands and displaying responses, and a tablet with pen for indicating positions. It also includes two devices which are not shown: a hard copy unit for reproducing displays on paper and a modem for communicating over telephone lines

applications-oriented operations on the type (e.g., computing a plausible three-dimensional conformation of a molecule).

PROPHET permits access to data types on two different levels. First, there is a large set of English-like *commands* to specify the most common kinds of operations. By using commands in a step-by-step manner, a user can perform non-trivial manipulations without knowledge either of programming or of the internal structure of the data types being used. Commands can be abbreviated; many spelling errors are corrected, and the terminal's tablet can be used to indicate arguments graphically (this is most convenient when the argument is a part of a larger entity such as a point on a graph).

It is also possible to write procedures which access the basic components of data types through *programming statements* modeled upon those of PL/I. A programming user can arbitrarily combine commands (which deal with data types as a whole) and PL/I statements (which can deal with their parts) in order to carry out a task. The existence of system-wide representations for entities, and the ability to intermix commands and programming statements freely, make it relatively easy for one user to use another's algorithms in his work.

A PROPHET example

To provide a better picture of PROPHET use, and to serve as a concrete reference point for discussion of implementation, a sample session with PROPHET is represented in Figures 4(a) through 4(e). The pharmacological content of the example is simplistic; it does, however, suggest how PROPHET's data-types, commands, and statements might be used to explore a problem.

The figures show the appearance of the screen at different times throughout the session. The text at the top of each figure is a record of the last few lines typed by the user (underlined) and by the system. The pictures in the center of the screen are generated by the system in response to the user's actions.

The user first logs onto the system and is told that he has several saved tables (structured data files) which he created in earlier sessions. He asks for one of them to be displayed; it appears as a matrix-like arrangement of cells. In Figure 4(a), the first 12 rows of a larger (21-row) table are displayed. The table contains one row for each of a number of amino acid derivatives, specifying their biological activities, common names, and molecular structures. The table was originally created by specifying the structure of the table (the names and data types of the columns) and then filling in the data (typing the numbers and texts and drawing sketches of the molecules). Tables are the primary means in PROPHET of entering, organizing, saving, and retrieving data.

In Figure 4(b), the user retrieves a subset of the entries by making up a new table containing only those rows which represent substrates with high activity. He causes this to be displayed in place of the original display with the EAD command (EAD is a system abbreviation for the common sequence ERASE ALL; DISPLAY). To obtain a clue about the relation of the structure of the molecules to their activity, he requests a conformational model of the most active variant by typing the COMPUTE MODEL command and pointing (with the tablet's pen) to the table cell containing the structure of that molecule (column 4 row 1). He requests that this model be displayed (Figure 4(b)) and repeats this operation for a second molecule.

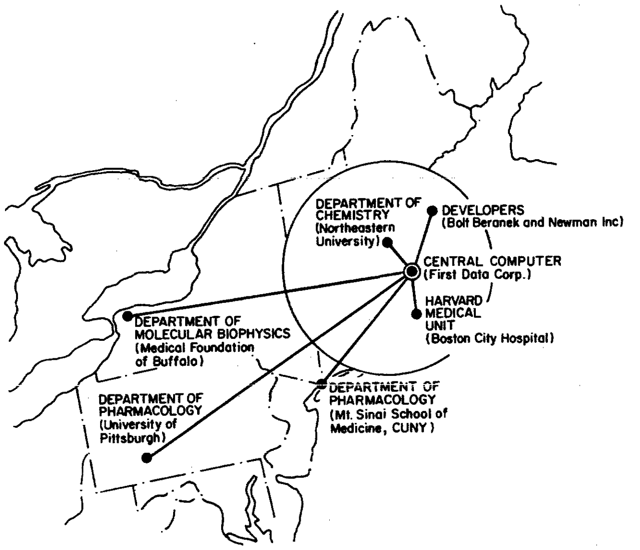

DEPARTMENT OF CHEMISTRY (Northeastern University)
DEVELOPERS (Bolt Beranek and Newman Inc)
CENTRAL COMPUTER (First Data Corp.)
HARVARD MEDICAL UNIT (Boston City Hospital)
DEPARTMENT OF MOLECULAR BIOPHYSICS (Medical Foundation of Buffalo)
DEPARTMENT OF PHARMACOLOGY (University of Pittsburgh)
DEPARTMENT OF PHARMACOLOGY (Mt. Sinai School of Medicine, CUNY)

Figure 2—PROPHET user map (October 1973)

INSTITUTION	USERS	INVESTIGATIONS
Department of Pharmacology University of Pittsburgh Medical School	Anderson, M.D., Ph.D. Somani, Ph.D.	Study of the interactions of uricosuric drugs. Mathematical modeling of enzyme systems near equilibrium conditions. Metabolism of neostigmine in isolated perfused rat liver. Experimental and simulation studies of the pharmacokinetics of secobarbital. Correlation of drug metabolite information in various animal species.
	Werner, M.D. Kulics, Ph.D. Fierst, Ph.D.	Investigation of the neuropharmacology of hallucinogens and narcotic analgesics by measuring single neuron activities in primates in response to tactile stimuli in various control, behavioral and drug states.
	Connamacher, Ph.D.	Experimental studies of the mechanism and locus of a phenotypic resistance to tetracycline. Study of the effects of tetracycline and selected inorganic ions on bacterial protein synthesis.
Harvard Medical Unit Thorndike Memorial Laboratory Boston City Hospital	O'Donnell, M.D. Blackburn, M.D. Feeney	Metabolic studies of sepsis and shock in pigs and humans. Studies of parenteral alimentation in humans and dogs. Liver metabolism studies.
	Silva, M.D.	Study of potassium metabolism in the isolated kidney. Study of Na-K-ATPase activity; the effect of different mineralocorticoids and relation to oxygen consumption.
	Speizer, M.D.	Epidemiological studies of exposure to automobile exhaust (initial screening of lead values in Turnpike Tunnel subjects).
	McGowan, M.D.	Demographic study of hospital admissions to obtain denominator data for the incidence of bacteremia and for cost-benefit analysis of bacteriological cultures.
	Wagner, M.D. Kane, M.D.	Effect of the manipulation of cardiovascular compliance data in animal subjects.
	Ransil, M.D., Ph.D. Auty, Ph.D.	Study of systolic time intervals following exercise. Experimental and mathematical investigation of insulin clearance in renal insufficiency. Standardization and use of amino acid analysis and gas chromatography in the clinical laboratory.
Department of Pharmacology Mt. Sinai School of Medicine City University of New York	Green, Ph.D. Johnson, Ph.D.	Correlation of physico-chemical properties of compounds with their biological activity, particularly: quantities computed from M.O. wavefunctions, Hansch π coefficient, and Taft σ parameter.
	Glick, Ph.D.	Study of behavioral effects in rats of certain drugs.
Department of Molecular Biophysics Medical Foundation of Buffalo	Duax, Ph.D. Rohrer, Ph.D. Weeks, Ph.D.	X-ray crystallographic studies of steroid and thyroid hormones. Structural-functional relationships, particularly: unusual Cotton effects in a series of androstanes, the relationship between A-ring conformation and glucocorticoid activity in a series of cortisol derivatives, and analyses of conformational transmission through the rings.
Departments of Chemistry and Medicinal Chemistry Northeastern University	Clagett, Ph.D. Kier, Ph.D. Aldrich, Ph.D. Lynch Mathers	Synthesis and study of ribosylureas as potential antimetabolites. Study of singlet oxygen-nucleoside reactions. Mathematical modeling of drug-receptor site interactions.
	Jankowski, Ph.D. Freiberg, Ph.D.	Study of the interaction of heavy metals with the environment and the transport of heavy metals through the first steps of the food chain.

Figure 3—Research investigations using PROPHET (October 1973)

```
GOOD MORNING, CHARLOTTE
YOU HAVE 10 PRIVATE TABLES
YOU HAVE 1 PUBLIC TABLE
THIS IS PROPHET 14
DISPLAY SYNTHETASE( )( )
```

SYNTHETASE 21 R 4 C	1 TYPE OF ACTIVITY	2 RELATIVE ACTIVITY	3 COMMON NAME	4 COMPOUND
1	SUBSTRATE	100.	L-GLUTAMATE	$C_5H_9NO_4$
2	SUBSTRATE	27.	D-GLUTAMATE	$C_5H_9NO_4$
3	SUBSTRATE	27.	THREO-3-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$
4	SUBSTRATE	2.	THREO-3-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$
5	SUBSTRATE	15.	ERYTHRO-3-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$
6	SUBSTRATE	11.	ERYTHRO-3-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$
7	SUBSTRATE	29.	CIS-L-CYCLOGLUTAMATE	$C_8H_{13}NO_4$
8	SUBSTRATE	0.	CIS-D-CYCLOGLUTAMATE	$C_8H_{13}NO_4$
9	SUBSTRATE	2.2	THREO-2-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$
10	SUBSTRATE	0.	THREO-2-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$
11	SUBSTRATE	0.	ERYTHRO-2-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$
12	SUBSTRATE	0.	ERYTHRO-2-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$

Figure 4(a)

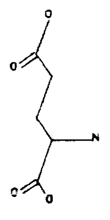
Hypothesizing that the differences in activity might be partially accounted for by the distances between the nitrogen and the furthest oxygen, the user decides to write a procedure to compute this distances and to enter them in the table as a new column. He types the MAKE PROCEDURE command, which causes him to enter a text editor to compose the procedure.

Figure 4(c) shows the completed procedure definition. It adds a column to the table, then cycles through the rows of the table. For each row, it retrieves the molecular structure, computes a three-dimensional model of the molecule, and cycles through the atoms to compute the maximum distance between the nitrogen and an oxygen; it then enters this value in the new column. The procedure uses a combination of commands and PL/PROPHET statements, including calls to system routines. As written, it depends heavily on characteristics of the particular molecules in the table, each of which has one nitrogen and one or more oxygens. In Figure 4(d) the user exits from the editor, compiles the procedure, calls it, and displays the table.

To see any structure/activity pattern more clearly, the user makes a graph (Figure 4(e)) from the activity column and the distance column. He fits a straight line to the graph and

```
MAKE TABLE TT FROM SYNTHETASE FOR TYPE='SUBSTRATE' AND RELATIVE>2(
READ TT( )COMPUTE MODEL OF ( )D18 ( THREED( )COMPUTE MODEL OF (
)D18 ( THREED( )MAKE PROCEDURE ONDIST( )
```

TT 8 R 4 C	1 TYPE OF ACTIVITY	2 RELATIVE ACTIVITY	3 COMMON NAME	4 COMPOUND
1	SUBSTRATE	100.	L-GLUTAMATE	$C_5H_9NO_4$
2	SUBSTRATE	27.	D-GLUTAMATE	$C_5H_9NO_4$
3	SUBSTRATE	27.	THREO-3-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$
4	SUBSTRATE	15.	ERYTHRO-3-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$
5	SUBSTRATE	11.	ERYTHRO-3-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$
6	SUBSTRATE	29.	CIS-L-CYCLOGLUTAMATE	$C_8H_{13}NO_4$
7	SUBSTRATE	2.2	THREO-2-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$
8	SUBSTRATE	75.	2-GLUTAMATE	$C_5H_9NO_4$



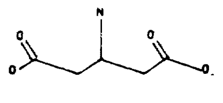


Figure 4(b)

```
PROCEDURE(T,C);

DECLARE  T  TABLE, M  MOLECULE;
DECLARE  (C,NC,R,NR,A,AA)  FIXED;
DECLARE  MAXD  FLOAT;

  CALL ADDCOLUMN(T,'O-N DISTANCE','FLOAT');
  CALL ROWSIZE(T,NR);  CALL COLSIZE(T,NC);

  DO  R=1  TO  NR;
    SET M TO COL C ROW R OF T;
    COMPUTE MODEL OF M;
    MAXD=0.;

    DO A=1 TO NUMBER OF ATOMS IN M;
      IF  M.ATOMS[A].TYPE='N'
        THEN DO AA=1 TO NUMBER OF ATOMS IN M;
                IF  M.ATOMS[AA].TYPE='O'
                  THEN  MAXD=MAX(MAXD,
        DISTANCE  M.ATOMS[A],M.ATOMS[AA]);
              END;
    END;

    IF  MAXD=0.0
      THEN ERROR CONTINUE,
        'MOLECULE IN ROW '.CTEXT(R).
        ' IS MISSING NITROGENS OR OXYGENS';

    SET COL NC ROW R OF T TO MAXD;
    SET COL C ROW R OF T TO M;
  END;

  RETURN;
END;
```

Figure 4(c)

```
COMPILE? YES( [COMPILING...DONE] )CALL ONDIST(TT,4)( )READ TT(
```

TT 8 R 5 C	1 TYPE OF ACTIVITY	2 RELATIVE ACTIVITY	3 COMMON NAME	4 COMPOUND	5 O-N DISTANCE
1	SUBSTRATE	100.	L-GLUTAMATE	$C_5H_9NO_4$	3.33
2	SUBSTRATE	27.	D-GLUTAMATE	$C_5H_9NO_4$	3.01
3	SUBSTRATE	27.	THREO-3-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$	4.21
4	SUBSTRATE	15.	ERYTHRO-3-METHYL-L-GLUTAMATE	$C_6H_{11}NO_4$	3.09
5	SUBSTRATE	11.	ERYTHRO-3-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$	2.9
6	SUBSTRATE	29.	CIS-L-CYCLOGLUTAMATE	$C_8H_{13}NO_4$	3.04
7	SUBSTRATE	2.2	THREO-2-METHYL-D-GLUTAMATE	$C_6H_{11}NO_4$	3.4
8	SUBSTRATE	75.	2-GLUTAMATE	$C_5H_9NO_4$	3.58

Figure 4(d)

displays it. The line-fitting routine warns him that the linear relation computed is unlikely to be significant. The user logs off, saving his work area (core image) so that he can continue from the same point at a later time.

Software Structure

The programs which comprise PROPHET and which support the kinds of capabilities shown in the example are extensive (currently more than a quarter-million machine instructions). These programs are not homogeneous but are divided into distinct layers of software which transform the capabilities of the computer to the form which PROPHET users see. These layers are a time-sharing operating system, an extensible base language, a set of display and table handling utilities, a body of applications programs and commands, and a library of user-written procedures.

As shown in Figure 5, each layer extends the preceding layer to produce a new computing environment. The time-sharing monitor builds on the bare computer to create a "user machine," in which many users can operate concurrently without interfering with each other and in which real resources are presented as virtual resources which are general, easy to use, and protected from other users. The extensible base language provides high-level programming statements which facilitate computation, the definition of new data types, and the addition of syntax for new statements. It parses incoming statements, compiles machine instructions, and allocates resources within the user area provided by the operating system. The display and table utilities, using both machine language and the extensible language, implement general schemes for handling the display and the filing system. The applications programs use high-level procedures, new data types, and commands to present an environment for the PROPHET user in which he can deal directly with

```
)MAKE GRAPH G FROM TT AS COL 5 VS COL 2(
)FIT LINE TO G(DO YOU WANT A FIT FORCED THROUGH ORIGIN ?NO(DO YOU WANT -
A CONFIDENCE LEVEL ON THE SLOPE ?YES(DO YOU WANT EMPTY Y VALUES FILLED 1-
N ?NO( USING 8 DATA POINTS
THE CORRELATION COEFFICIENT IS R = .1994188
THE BEST STRAIGHT LINE IS:
Y = 15.86299*X + -16.89012
DO YOU WANT TO ADD THAT FUNCTION AS A NEW CURVE IN YOUR GRAPH ?YES(
THE SLOPE HAS A CONFIDENCE LEVEL OF LESS THAN 50. PER CENT
)EAD G( LOGOUT( SAVE WORKAREA? YES(
```

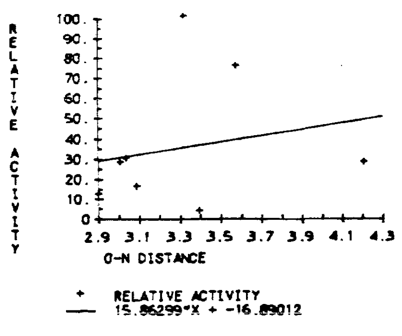


Figure 4(e)

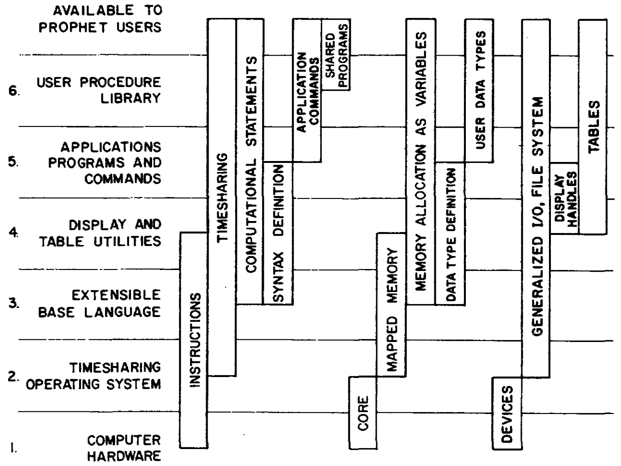

AVAILABLE TO PROPHET USERS
6. USER PROCEDURE LIBRARY
5. APPLICATIONS PROGRAMS AND COMMANDS
4. DISPLAY AND TABLE UTILITIES
3. EXTENSIBLE BASE LANGUAGE
2. TIMESHARING OPERATING SYSTEM
1. COMPUTER HARDWARE
INSTRUCTIONS
TIMESHARING
COMPUTATIONAL STATEMENTS
SYNTAX DEFINITION
APPLICATION COMMANDS
SHARED PROGRAMS
CORE
MAPPED MEMORY
MEMORY ALLOCATION AS VARIABLES
DATA TYPE DEFINITION
USER DATA TYPES
DEVICES
GENERALIZED I/O, FILE SYSTEM
DISPLAY HANDLES
TABLES

Figure 5—The numbered levels correspond to the layers of PROPHET hardware and software. The vertical bars represent each of the major computer capabilities within the system. Each capability is provided by the layer at the bottom of the vertical bar to each of the higher layers that the bar passes through. (For example, the bar labeled "SYNTAX DEFINITION" indicates that the base language provides this capability of defining new command syntax for use in constructing the utilities and the applications commands; on the other hand, creators of the user procedure library or general PROPHET users do not have access to this capability

entities and applications related to his research. Finally, the library of user-written programs allows users to build on the basic system capabilities to handle new applications.

The use of distinct layers of software has had several important benefits. It has made it possible to write, maintain, test, and document sections of the software independently, without requiring that each implementer have a comprehensive knowledge of the system. The creation of a succession of documented, tested programming environments means that new operations can be programmed quite simply (e.g., a programmer writing a procedure to twist a molecule can concentrate on the application without explicitly providing for multi-user use of the procedure, allocation of storage for data structures, or displaying or filing the altered molecule). Because of the isolation of functions into layers, there have been relatively few bugs related to complex interactions between parts of the system. Development of separate layers has also been valuable in reducing the impact of basic system changes (for example, a change to a new operating system affected only the base language and the display and table utilities).

A drawback of the layered approach is a cost in efficiency. For example, use of a generalized filing system is slower (perhaps by a factor of 2) than would be direct access to disks; and applications programs written in the base language run more slowly (by an average factor of 5) than if they were written in assembly language.

Despite this cost in efficiency, several critical constraints precluded such alternatives as writing the whole system in assembly language. Specifically, given the three constraints

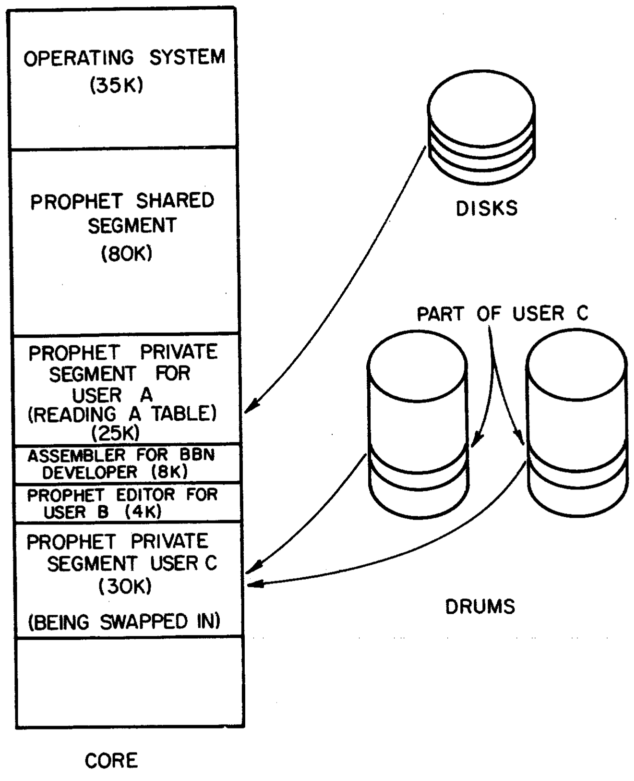

OPERATING SYSTEM (35K)
PROPHET SHARED SEGMENT (80K)
PROPHET PRIVATE SEGMENT FOR USER A (READING A TABLE) (25K)
ASSEMBLER FOR BBN DEVELOPER (8K)
PROPHET EDITOR FOR USER B (4K)
PROPHET PRIVATE SEGMENT USER C (30K) (BEING SWAPPED IN)
CORE
DISKS
PART OF USER C
DRUMS

Figure 6—Memory snapshot of the PROPHET system

of developing a useful service system within two years, of allowing for modifications and extensions of the original design as experience with users required, and of providing an easy language for user programming, the layered approach has proved invaluable if not essential.

PROPHET IMPLEMENTATION

As shown in Figure 5, PROPHET has been implemented in six distinct levels. Each level is discussed in turn.

Level 1: Computer hardware

PROPHET runs on a large Digital Equipment Corporation PDP-10 system. The PDP-10 was selected in 1969 primarily because of its moderate price and working time-sharing system. The configuration has 192K 36-bit words of core memory and two separate swapping-drum systems (drums and controllers) to help handle the large core images used in PROPHET. The system has six disk-pack drives (a total of 150 million characters of on-line storage) as well as magnetic tapes for backup and for exchange of programs and data with other systems.

The system currently supports two kinds of terminal: Teletypes for BBN system developers and Computek 400 display terminals for users. The Computek (Figure 1) was selected in 1970 because it supports the minimal input and output functions of PROPHET at a low price and in a form which places low computational and transmission demands on the central computer. The display is based on a storage tube, which produces a stable, highly readable display of an arbitrarily complex picture without being refreshed from the main computer. The principal defects of the display are its inability to make partial deletions (the entire screen must be erased and the picture retransmitted) and the low brightness of the screen (particularly of the non-storing cursor). The terminal includes a tablet which has been modified so that "tracking" (moving the cursor on the screen to correspond to the current position of the tablet's pen) is a purely local operation, and data is sent to the computer only when the pen is pressed down to indicate a point.

Level 2: Operating System

The operating system used for the service PROPHET system is a version of the DEC system-10 level D monitor. The system also runs on, and was developed on, the BBN Tenex monitor. The major function of the operating system is creation of a multi-user environment. The system does this by allowing programs to be written as if they were to be run alone, then dynamically assigning them real core and the CPU for short periods of time in order to run. Figure 6 shows what might be the contents of the main memory at an instant of time with several PROPHET users active. Four operations are happening in parallel: User A is reading data from the disk; User B is running his program; one part of User C is being read in from one drum, while the other part is being read from the second drum. Also, the operating system provides for several programs to share a single copy of common routines. The main portion of PROPHET is shared in this way.

The operating system organizes storage on disks into a file structure, in which a user program can create, read, write, and update named files. This frees a program of the necessity of knowing exactly where on the physical device its data is placed (this can be complicated in view of shared use of the device, "bad spots" on the disk, the potential of data moving when the disk undergoes maintenance, and so forth). The DEC file system also allows protection codes to be assigned to files so that access rights may vary for the creator, for his group of users, and for the system as a whole.

A few modifications have been made to tailor the monitor to PROPHET. The most important additions have been programs to support the dual-drum swapping hardware and to provide special echoing conventions for the display terminal.

Level 3: Base language

The base language serves as the programming language for approximately three-quarters of the PROPHET software. Because of its central importance in the system, and because it is one of the more novel portions of the system from a computer science point of view, the base language level is discussed here in some detail.

The base language is called PARSEC (for PARSer and Extensible Compiler). Many of the approaches taken in designing PARSEC extensibility were patterned after the language Proteus, developed by Dr. James Bell at Stanford. This language was selected as a starting point after a survey in 1968 of the major extensible language efforts then under way. In implementing PARSEC, techniques and features were borrowed from other language efforts.[7,9,10] The computational statements are taken from PL/I, omitting the fancier mechanisms for storage control (dynamic, static, own), input/output (list, data), and run control (tasks, ON statements). The internal design of PARSEC is discussed in detail in *An Overview of PARSEC Implementation.*[5]

The overall function of the base language level of PROPHET is the analysis and execution of several kinds of program statements. These statements (a) perform computation (arithmetic, accessing data structure, iteration, branching, subroutine calling, etc.), (b) define new data types as aggregates of existing types, (c) define syntax transformations to parse new statement formats, and (d) control the operating environment. Each of these functions is discussed below.

(a) Computational statements

The computational statements of the base language are based on a subset of PL/I. They include declarations, assignment, full IFs and DOs, block structure, and procedures. Operations are defined on fixed and floating point numbers (arithmetic), Boolean variables (logic), text (concatenation, location, extraction), and bit strings (bit by bit logic). Multidimensional arrays of any type can be defined.

All variables are allocated dynamically when they are used. Text strings and arrays are completely variable in length. Assignment is pointer-oriented; it is defined as copying a pointer to the result of the right-hand side into the location referenced by the left-hand side. This allows complex (even circular) structures to be built and manipulated, and avoids wasteful reallocation in system handling of PROPHET data types. A unique pointer value "empty" is used to represent uninitialized variables; use of this value in an operation results in an error message, which helps to detect failures to set things up correctly.

The computational language is intended to be flexible: mixed mode operations are allowed, undeclared variables are implicitly declared, expressions are permitted wherever sensible, and nesting of IFs, DOs, and expressions is allowed to any depth. The result is a language which is designed to be extremely easy to use; when misused, however, it can incur noticeable inefficiencies. For example, while variable-length arrays allow a programmer to write and run procedures using arrays without the bother of dimensioning, a program can cause repeated reallocation of the array if it expands the array one element at a time. With care, inefficiencies of this kind can be avoided in system routines, while allowing PROPHET users to write simple programs without efficiency-motivated restrictions.

```
COMPLEX NUMBERS

   DECLARE COMPLEX PATT!                      make a template called
                                                "COMPLEX"
   COMPLEX="<RP:FLOAT> + <IP:FLOAT>I"!        define real and imaginary
                                                fields

   DECLARE X COMPLEX!                         declare a complex variable
   X.RP=1!                                    set the real part
   X.IP=2!                                    set the imaginary part

   TYPE X!                                    type the value
   1. + 2.I
   TYPE X.IP!                                 type a field
   2.


LISP LISTS

   DECLARE LISPLIST PATT!                     make a template called
                                                "LISPLIST"
   LISPLIST="(<HEAD:ANY> . <TAIL:LISPLIST>)"! define fields

   DECLARE LL LISPLIST!                       declare a LISPLIST
   LL.HEAD.='A'!                              set head
   LL.TAIL.HEAD='B'!                          set second element's head

   TYPE LL!                                   type value
   (A.(B.***))                                *** denotes an undefined
                                                field

   LL.TAIL.TAIL=LL!                           make list circular
   TYPE LL!
   (A.(B.$$$))                                $$$ denotes circularity
                                                (to prevent indefinite
                                                typeout)
```

Figure 7—Data type definition and use. The first section defines a complex number, previously unknown to the system, as an entity with a real and an imaginary part. The second section defines a list element like that used in LISP

(b) Data type definition

New data types can be defined in the base language as an ordered set of elements of other data types. Figure 7 shows how two compound types can be defined and used. To make a new data type, a template is formed which lists the elements, assigning a name and a data type to each. This template can then be used to declare instances of the data type. An instance of a compound data type can be treated in much the same way as one of the basic types, (*viz.*, an integer or a text string): it can be declared, be used as an argument to procedures, have operations defined on it, be typed out, and so forth. The components of a compound data type can be accessed through dot-notation, in a manner similar to accessing structures in PL/I. Components can in turn be compound types (the lower levels can be accessed through further levels of dot-notation).

The utility of compound data types can be underestimated by those who have not worked with them. Although many of the same basic capabilities can be obtained by using arrays or simple variables to represent the different parts of a compound object, compound types have important advantages when either the elements of the representation or the kind of permissible operations change. Because structures are handled as single variables, calling sequences never have to change when components are added or deleted. Components are referenced by name, not by location within the structure, so references need not be changed when unrelated

```
COMPUTE MODEL COMMAND

COMPUTE:  %MACRO;                               start macro definition

          %RULE "<>MODEL OF"=                   throw away noise words
                "<>"!
                -!
                -;                              to next rule in any case

          %RULE "<> <M:MOLECULE>"=              if.user typed a variable of
                "<>CALL CM(<M:MOLECULE>)"!        type molecule, set up
                                                  procedure call to do
                                                  computation
                -!
                RETURN;
          %RULE "<> <T:TABLETINDICATION>"=      if user pointed to a
                "<>CALL CM(<M:MOLECULE>)!"!       molecule on the screen
                                                  call compile-time routine
                                                  to find it and set up call

                BEGIN;
                  CALL FINDVAR(M,T,UDL);
                  IF EMPTY(M) THEN FAIL;
                END!
                RETURN;
          FAIL;                                 announce an error if the
          END;                                    argument was not of type
                                                  molecule and the user did
                                                  not point to a molecule
```

Figure 8—Syntax definition of the COMPUTE MODEL command. The definition consists of a set of syntax rules which are executed when the system encounters the keyword "COMPUTE". The first rule discards the words "MODEL OF"; the second rule compiles a program call for the case in which a user named a molecule; and the third rule compiles a call for the case in which the user pointed to a molecule on the screen

components are redefined. Perhaps the most important advantage in a large system like PROPHET is that programs using compound data types are easy to write and to read.

(c) Syntax definition

The base language also allows the definition of new statement syntax on an equal basis with the original syntax of the language without changing the compiler. The base system contains a set of machine-language routines to process user input according to a set of syntax statements which have been supplied to it. The syntax statements supplied to the compiler fully define the language which it is to implement. The base language itself is constructed from lists of syntax statements which define the formats for the computational statements, the data-type definition facilities, and the syntax definition facilities. New syntax can be added to extend the language or even to redefine the original base language; this is in fact used in the process of loading the base language to bootstrap the full base language from a simpler initial language.

The syntax interpretation process starts by breaking the user's input down into a series of tokens which are represented as name-type-value triples. A token can be a variable name, a special character, or a constant. A msater list of syntax statements is then called to interpret the set of tokens and to transform it to collections of progressively "simpler" forms which can ultimately be represented in machine language. For example, a command input by a user as "TYPE 1,2" is transformed to a longer equivalent form of simpler components "DO; OUTPUT 1; OUTPUT 2; END" in the process of compiling it into machine language.

Most of the work of a transformation is performed by a syntax statement called a rule. A rule performs a matching/replacement function similar to that of an AMBIT statement.[8] It consists of five parts: an input pattern, an output pattern, a program, a pointer to an arbitrary syntax statement, and a pointer to the next statement in the list. The input and output patterns each consist of a template set of tokens (which can be named so that both templates can refer to the same token). When a rule is executed, the input pattern is matched against the set of tokens derived from the user's input. Matches can be against specific sets of characters or against any token of a given data type.

If a match is found, the output pattern is inserted into the set of tokens at the point of match, and the program part of the rule is called. This program can be any statement in the base language, such as a block or procedure call. The program can access the matched tokens to perform further checking or to compile instructions. The program can terminate by announcing success or failure. If it fails, the *output* pattern is removed from the set of tokens and control passes to the next syntax statement in the list. If it succeeds, the tokens matched by the *input* pattern are removed and control branches to the success pointer specified in the rule. Other syntax statements exist to control flow through the statement lists. Figure 8 is an example of syntax which defines the COMPUTE MODEL command.

(d) Control statement

A fourth class of PARSEC statement is used to control the operation of the base system in various ways. The most powerful of these statements is the RUN statement. RUN saves the current state of PROPHET and causes a specified PDP-10 program to be executed. This program may be written in any language available on the PDP-10 (assembler, FORTRAN, LISP, etc.), and be essentially the same as a stand-alone version of the program (the only necessary change is insertion of a call to a machine-language library program to return to PROPHET). It is possible to use the RUN command within a PROPHET procedure, and to have control return to the correct point in procedure execution. Files can be used to communicate arguments and results between PROPHET and the program being run. The RUN statement has been valuable in implementation of sections of the system which require very fast response (the text editor is a small, easily swapped program—written in assembly language—executed through RUN) and in adaptation of large existing programs to interface to PROPHET (for example, the ORTEP molecule display program—written in FORTRAN—for crystallographers).

Level 4: Display and table utilities

The levels composed of the computer, the operating system, and the base language are still a Teletype-oriented system, with access to files only on a do-it-yourself basis. The display and table utilities extend the system toward the PROPHET user environment by providing for displaying user data and filing it in a structured way. Because the specific user data types, and the representations selected for them, are not fixed, display and table-handling schemes were designed to be sufficiently general to survive additions and changes in user types without severe dislocation.

Displays

The overall purpose of the display utility is to provide a framework for managing the display terminal and for displaying arbitrary data types, and within that framework to provide specific routines to display tables, molecules, graphs, and statistical samples.

One set of utility routines performs the lowest level operations on the display terminal (e.g., draw a line, display a character, erase the screen, or activate the hard copier). Because the storage-tube display requires that the entire screen be erased and much of its contents be retransmitted to effect even a small change, the terminal-handling routines save the actual bytes sent to the terminal so that an individual display can be redrawn without the program's having to recompute how the display should be presented.

Another set of routines exists to present each data type in the way best suited to its meaning. For example, a table is displayed as a grid of cells, which may be "folded" to fit as much as possible on the screen, a molecule as a figure with lines for bonds and characters for atoms, a graph as a Cartesian plot with curves, labels, ticks, and so on. The routines which display an object access it in a read-only fashion. This safeguards the displayed object in the event of interruption of display, and permits the same object to be displayed in several ways at the same time.

Above these data-type-dependent routines, there is another set of display operations which is common to all data types. When an object is displayed, an instance of a special data type called a display-list is created consisting of pointers to the object, to a data-type-dependent entity which is used to store user-specified display options, to the coordinates of the area of the screen in which the object is displayed, and to the saved bytes from the last regeneration of the display on the screen. These display lists are linked together into a tree structure representing all objects currently being displayed. Control routines exist to manipulate the tree and to walk the tree to cause different sets of objects to be regenerated.

There is a second kind of operation, parallel to displaying, which requires specialized handling for each data type: correlating user tablet input with meaningful entities within the data type. Handling this operation is not as simple as it may appear at first because the meaning of pointing to a displayed element may vary according to the context. For example, in different operations, a user might wish pointing to an atom of a molecule to mean that atom (e.g., type the coordinates), the molecule as a whole (e.g., compute a model of that molecule), or the display of that molecule (e.g., move the display).

The finding operation is implemented by cycling through the current list of displays and pretending to redisplay them. Instead of drawing on the screen, the lowest level routines notice when lines or characters would have appeared near the coordinates where the user pointed. A stack is maintained during this process to represent the hierarchy of constructs currently being searched; from this stack routines can select which level they wish to deal with when a hit is detected (in the example above, the atom, the molecule, or the display).

Tables

The table-handling utility provides a means of storing and organizing data in a format which is familiar to users. The PROPHET table is patterned after the tabular formats used in scientific journals or in laboratory notebooks. It is displayed as a matrix-like structure with rows and columns; each column is constrained to contain instances of just one data type, but different columns can contain different types. The user specifies the number of columns and the types they should contain. Any data type may be specified for a particular column, including molecule, graph, and even table itself. A special kind of column can be defined which is filled by deriving values from the entries in other columns. For example, one column might be defined as containing the natural logarithms of the values in another column. Tables also permit such niceties as names for columns and (optionally) rows, notes, and the ability to limit the entries in a column to a user-specified list of permitted values.

A large number of user commands have been defined to manipulate tables. There are commands to create a new table, to make a table by selecting parts of one, to enter data in various ways, to access individual existing cells, and to display different sections of the table. Commands can also be used to retrieve information from tables according to specified criteria. In this context, tables are logically like record-oriented files, in which the rows are the records and the columns define the fields. It is possible to retrieve entries according to an arbitrary Boolean combination of criteria, and also to sort tables on the values in a set of columns.

Tables are implemented using the file structure provided by the operating system, one table per file. Although it involves slight inefficiencies, the use of the file structure has simplified implementation (allocation on the disk is trivial) and has greatly enhanced reliability. Use of the file structure also provides protection against unauthorized access, allows tables to be saved off-line and restored individually, and makes it possible for system implementers to access tables in a limited way from outside of PROPHET.

As a table command is processed, data are read from the table in rectangular arrays of entries (called windows); in no case must a program read an entire table into core, so that programs can deal with tables which are too large to fit in core. Cells containing variables which are large (such as

PROPHET Commands (listed alphabetically)

```
ABBREVIATE 'text' TO unused-name

       SHARABLE
ACCESS PUBLIC   TABLE table-name OF user-name AS unused-name

ADD COLUMNS TO table-name [FROM table-name [FOR relational-expression]]
ADD CURVES TO graph-name AS X-specification VS Y-specification
                            [,Y-specification,...]
ADD CURVES TO graph-name FROM graph-name
ADD ROWS TO table-name [FROM table-name]
CALL commandlist-name
CENTER ON element-indication - used for molecular models
CLEAN
      COLUMN
CLEAR ROW    number OF table-name
COMPLAIN
COMPUTE MODEL OF molecule-name
        GRAPHS
DEFAULT MOLECULES TO display option(s)
        TABLES
DELETE variable-name
DELETE ABBREVIATION abbreviation-name
       COLUMN
DELETE ROW    number OF table-name
DELETE CURVE curve-number OF graph-name
       [SHARABLE]
DELETE [PUBLIC  ] SAVED variable-name
DELETE WORKAREA [workarea-name]
DISPLAY variable-name [display option(s)] [position-indication]
DISPLAY graph-name [[NO]KEY] [[NO]LABELS] [[NO]WARNINGS]
                      [THREED]
DISPLAY molecule-name [SKETCH] [[NO]COMPLETE] [[NO]LABEL] [[NO]INDEX]
                      [STEREO]
                   [FULL    ]
DISPLAY table-name [HEADINGS] [[NO]FOLD] [[NO]ROWNAMES] [COLUMN[S]column-number(s)]
                   [QUICK   ]                           [ROW[S] row-number(s)]
DISPLAY text-name [LINE[S] line-number(s)]
DISPLAY COLUMN column-number ROW row-number OF table-name [display option(s)]
     text-name
EDIT commandlist-name
     procedure-name
ENTER COLUMN column-number(s) OF table-name
ENTER ROW row-number(s) OF table-name
      ALL
ERASE display-indication

FILLIN COLUMNS column-numbers] [ROWS row-numbers] OF table-name FROM
       [COLUMNS column-numbers] [ROWS row-numbers] OF table-name
    LINE          COLUMN column-number VS COLUMN column-number OF table-name
FIT POLYNOMIAL TO CURVE curve-number OF graph-name
FREEZE display-indication - used for molecular models
LIST ABBREVIATIONS
LIST VARIABLES [OF TYPE data-type] [WITH VALUES]
LOGIN
LOGOUT
MAKE COMMANDLIST unused-name
MAKE GRAPH unused-name [FROM table-name] [AS x-specification VS Y-specification
                                          [,Y-specification,...]]
MAKE MOLECULE unused-name [FROM molecule-name]
                               array-name
MAKE SAMPLE unused-name FROM   table-name COLUMN column-number
     [SHARABLE]                    [FROM table-name [FOR relational-expression]]
MAKE [PUBLIC  ] TABLE unused-name  [USING table-name                             ]
MAKE TEXT unused-name
MOVE element-indication position-indication
MOVE TO COLUMN column-number ROW row-number
MOVE TO LINE line-number
               X
ORIENT [ABOUT] Y bond-indication - used for molecular models
               Z
PRINT variable-name [display option(s)]
RENAME table-name TO unused-name
        [SHARABLE]
RESTORE [PUBLIC  ] saved-name [AS unused-name]
RESTORE WORKAREA [workarea-name]
     [SHARABLE]
SAVE [PUBLIC  ] variable-name [AS saved-name]
SAVE WORKAREA [workarea-name]
                         X
SCALE [scale-factor] [IN Y ] display-indication
                         Z
SCAN variable-name [display option(s)]
                         'user-name'
SEND MESSAGE 'text' TO   'PROJECT'
SET variable-name TO expression
                              column-number(s)    [ASCENDING ]
SORT table-name BY COLUMN[s]  column-heading(s)   [DESCENDING]
                      X
TURN angle [[AROUND]  Y ] display-indication
                      Z
TWIST angle [AROUND] atom-indication atom-indication - used for molecular models
TYPE variable-name
WHAT IS element-indication - used for the value of a data point in a graph
```

Figure 9—List of PROPHET commands (excluding the PL/I statements). Command arguments or modifiers listed in brackets are optional; those listed vertically indicate alternatives that are permitted. Lower case phrases describe the data-type of the argument expected; italics are used for comments

graphs or molecules) are not read into core unless they are *specifically* referenced, even if they lie within the rectangular window.

Level 5: Applications commands

The special PROPHET data types, procedures, and commands are intended to aid non-programming users in dealing with real problems, and to provide handles for expansion of the system by users who are programmers. Figure 9 gives a list of the commands available to the PROPHET users.

The existence of the two preceding levels (the base language and the display and table utilities) significantly reduces the level of effort involved in the design and construction of powerful applications tools. The data-type definition capability makes it possible to define a representation of an applications concept which can be accessed either as a whole or as a set of components. High level programs can be written to perform transformations on variables of the new type without concern for the realities of running (e.g., memory management, creation of machine code, and details of input/output and error handling). The new type can be displayed by adding appropriate displaying and searching routines to the display utility, and *nothing* need be done to allow it to be filed in tables. Finally, a set of commands, with arbitrary format and using graphic input, can be implemented through syntax definition statements.

Graphs

An example of a relatively simple data type is a 2-D data plot or graph (see example in Figure 4(e)). Several different relationships, based either on explicit data points or on functions, can be presented on the same graph. The user is allowed full control over presentation (specifying log or linear axes, labeling, tick distances, symbols used in plotting, ranges to be shown, titles, keys, and so forth). When an aspect of presentation is not specified by the user, a sensible default is provided.

A challenging aspect of graph implementation was the construction of display procedures to handle all possible combinations of specified and unspecified presentation options. For example, it is hard to select a default low value for an axis, in the case of a log axis with data values near zero, which simultaneously (a) includes all the data but does not leave much of the graph empty, (b) is a reasonable number which can be labeled in a few digits and (c) is an even multiple of some tick distance from the high value. In order to handle this type of problem, numerous heuristics had to be included in the graph-display package.

Molecules

Because many of the PROPHET users work with molecular structures, the molecule-handling commands are a central part of the system.

The representation selected for molecules is a structure containing an array of atoms, an array of bonds, and some information related to the molecule as a whole (Figure 10). Some fields are redundant and are included to make it easy to handle the structure in a variety of ways (e.g., iterating on each atom, handling all atoms connected to a given atom, or finding the atoms connected by a bond). To conserve storage space, some fields are not filled in until needed.

A user creates an instance of a molecule by drawing it or by changing an existing molecule. The drawing program uses the tablet to allow the user to point to light-button commands and to positions on the screen. We have found that the semi-static nature of the current storage-tube terminal is not a severe restriction on this kind of task.

A set of procedures computes a likely 3-D conformation of a user-sketched molecule by recognizing local conformations (rings, functional groups), and then putting the pieces of the molecule together using standard bond lengths and bond angles, modified by user-specified exceptions. Generating good conformations for rings is a very complex problem. Our current approach is to use a dictionary of rings, which is expanded at user request; our theory being that because users tend to work with small groups of related compounds, it is possible to limit the size of the dictionary. This approach has not been entirely satisfactory because the process of looking up rings is slow; users do not like having to find coordinates for their rings, and many rings have to be entered to cover minor variations in one basic ring structure.

```
MOLECULE

   FORMULA:TEXT . . . . . . . . . .chemical formula
   WEIGHT:FLOAT . . . . . . . . . .molecular weight
   ATOMS:ATOM ARRAY . . . . . . . .the molecule's atoms
   BONDS:BOND ARRAY . . . . . . . .the molecule's bonds
   SKETCH:BOOL. . . . . . . . . . .whether sketch drawn
   MODEL:BOOL . . . . . . . . . . .whether model computed

ATOM

   TYPE:TEXT. . . . . . . . . . . .chemical symbol
   CONN:ATOM ARRAY. . . . . . . . .pointers to neighboring atoms
   BONDS:BOND ARRAY . . . . . . . .pointers to connecting bonds
   LOCS:FLOAT ARRAY(2). . . . . . .user sketch coordinates
   POSITION:FLOAT ARRAY(3). . . . .conformation coordinates
   ANGLES:ANGLE ARRAY . . . . . . .user-set bond angles

BOND

   TYPE:TEXT. . . . . . . . . . . .double, resonant, etc.
   ATOM1:ATOM . . . . . . . . . . .pointer to first endpoint
   ATOM2:ATOM . . . . . . . . . . .pointer to second endpoint
   LENGTH:FLOAT . . . . . . . . . .user-set bond length
   DIHEDRAL:ANGLE . . . . . . . . .user-set dihedral angle

ANGLE

   BOND1:BOND . . . . . . . . . . .pointer to first bond
   BOND2:BOND . . . . . . . . . . .pointer to second bond
   ANG:FLOAT. . . . . . . . . . . .angle in degrees
```

Figure 10—Principal parts of the molecule data type

We are now considering the addition of a more sophisticated, energy-minimization program developed by Dr. Todd Wipke,[14] which promises to construct good approximations of most rings.

Another set of procedures has been written to match fragments of molecules against the contents of a column of type molecule in a table. This matching is employed both in user information-retrieval requests (as a criterion which can be specified for this kind of column) and in retrieving rings from the ring dictionary. The basic approach used is loosely based on the Sussenguth graph-matching algorithm;[16] it matches by using sets of atoms with like properties rather than by an atom-by-atom tree-matching scheme.

Work is now in progress to allow users to deal with interactions between several molecules, both graphically and by computing energy parameters, and to extend the conformation and substructure search capabilities.

Level 6: User procedure library

The users of the system are currently generating a layer on top of the basic PROPHET system composed of procedures they have written to handle their applications. Facilities exist for creating shared and public procedures, and a *Public Procedures Notebook*[17] has been started. Because of the RUN statement in the base language, it is not always necessary for an existing procedure to be recoded to be used from PROPHET; a FORTRAN application program can, for example, be run from a PARSEC procedure which writes out a file of arguments, runs the program, and reads the results back in. Currently, a small set of user-created library programs exist (to manipulate graphs, to perform matrix manipulations, and to solve sets of linear equations). This library should facilitate one of the important goals of PROPHET—the exchange of data and programs among biomedical scientists working on related investigations.

BIBLIOGRAPHY

1. *The PROPHET System—A Final Report on Phase I of the Design Effort for the Chemical/Biological Information-Handling Program,* Bolt Beranek and Newman, Inc. (June 1970).
2. *User's Manual for the PROPHET System,* Bolt Beranek and Newman, Inc. (last revised January 1974).
3. Raub, William F., "The Role of Specialized Data Structures in Computer-Based Management, Analysis, and Communication of Pharmacological Information," National Institutes of Health (1972).
4. *PARSEC Manual,* Bolt Beranek and Newman, Inc. (29 September 1972).
5. *An Overview of PARSEC Implementation,* Bolt Beranek and Newman, Inc. (September 1972).
6. Bell, J. R. *The Design of a Minimal Expandable Computer Language,* Stanford University Ph.D. Dissertation (December 1968).
7. Christensen, C., and C. J. Shaw (Eds.), *Proceedings of the Extensible Languages Symposium,* SIGPLAN Notices 4 (August 1969).
8. Christensen, C., "A Language for Symbol Manipulation," *Comm ACM* 9 (August 1966), 570-623.
9. Irons, E. T., "Experience with an Extensible Language," *Comm. ACM 13* (January 1970), 31-40.

10. Cheatham, T. E., and K. Sattley, "Syntax Directed Compiling," *Proc. AFIPS FJCC*, 24 (1964), 31-57.
11. Sutherland, I. E., "Sketchpad, A Man-Machine Graphical Communication System," *Proc. AFIPS, 1963 SJCC*, Spartan Books, Baltimore, Md.
12. Levinthal, C., "Molecular Model-Building by Computer," *Sci. Amer.*, 214: 42 (1966).
13. Corey, E. J. and W. T. Wipke, "Computer-Assisted Design of Complex Organic Syntheses," *Science*, 166: 3902 (1969).
14. Wipke, W. T., *et al.* "Computer Construction of Stereochemically Correct, Three-Dimensional Molecular Models from Two-Dimensional Structural Diagrams or from Connection Tables," Princeton University (paper in preparation).
15. Barry, C. D., R. A. Ellis, S. M. Graessner, and G. R. Marshall, "Molecular Modeling with a Small Computer: an Introduction and Results of Initial Use," *Computer Systems Laboratory Technical Memo. No. 96*. St. Louis: Washington University (1970).
16. Sussenguth, E. H., "A Graph-Theoretic Algorithm for Matching Chemical Structures," *J. Chem. Doc.*, 5: 36 (1965).
17. *Public Procedures Notebook*, Bolt Beranek and Newman, Inc. (April 1973).

Applications of the PROPHET system in correlating crystallographic structural data with biological information*

by CHARLES M. WEEKS, VIVIAN CODY, STEPHEN POKRYWIECKI, DOUGLAS C. ROHRER and WILLIAM L. DUAX

Medical Foundation of Buffalo
Buffalo, New York

The goal of the Molecular Biophysics Department at the Medical Foundation of Buffalo is to establish relationships between the physical structures of molecules and their biological activities. At present, the steroid and thyroid hormones, their derivatives, analogs, and inhibitors are the materials of major interest in this research, but work on other groups of biologically active molecules is anticipated or in progress. The PROPHET system** provides a powerful medium for the assembly, storage, and analysis of both structural and biological data. Correlative studies of these two types of data are particularly important because they may lead to an understanding of the molecular level mechanisms of action of hormones, drugs, antibiotics, and other biological substances. The PROPHET system is well suited for this type of analysis because it permits communication and interaction among scientists who are experts on various phases of molecular biology.

The investigation of structural-functional relationships as it is being pursued at the Medical Foundation may be divided into the following stages: (1) The basic structural information must be collected and stored within the PROPHET system. X-ray crystal structure determinations in this and other laboratories have provided a large volume of molecular structural information in the form of atomic coordinates. Any geometric parameter of interest such as interatomic distances, bond angles, torsional angles, and deviations from least-squares planes can be calculated from these coordinates. The coordinates for 200 steroids, 50 aromatic amino acids, and 25 thyroactive compounds have been stored within the PROPHET system. (2) The geometrical data for groups of similar substances are examined for correlations between structural features and chemical or biological properties. (3) An attempt is made to explain the mechanism of action of the molecules of interest in terms of any observed correlations. (4) Speculation is made concerning what changes in the molecular structure may be possible and what effects these changes will have on the biological action of the molecule. The present paper will illustrate some of the ways in which the PROPHET system has been used to compare conformational features of related molecules and to correlate the structural differences with differences in chemical and biological properties.

The atomic coordinates and related information for each molecule in the data bank are stored in disk files where they may be accessed either by FORTRAN programs or PROPHET procedures. A PROPHET variable of type MOLECULE is also created for each structure. A pictorial representation of a molecule may be created either by sketching a chemical diagram on the tablet or by executing a PROPHET procedure which uses the crystallographic coordinate file to determine the positions of the atoms and chemical bonds. Both methods generate a connection table, and some miscellaneous information such as the molecular weight and percent composition is also computed. PROPHET can make the equivalent of a three-dimensional Dreiding model for acyclic molecules and for simple cyclic compounds containing rings which are included in its ring dictionary. Figure 1 shows sketches of the steroid hormone cortisol and the thyroid hormone triiodothyronine (T3). Figure 2 illustrates the three-dimensional model of T3 computed from the sketch as well as a view of the actual crystallographically observed structure. At present, a three-dimensional model cannot be computed for the fused ring system comprising the steroid nucleus. For obvious reasons, the sketching feature is used only when experimental coordinates are not available or for drawing a molecular fragment which is to be used in a substructure search.

The TABLE and GRAPH functions of the PROPHET system may be used to analyze information derived from the coordinates and to arrange this information so that statistics may be computed in a convenient fashion. One conformational feature of the thyroactive compounds which was analyzed in this way is the disposition of the planar aromatic rings about the connecting oxygen atom. The gross conformational changes in the molecule which accompany rotations about

* This research is supported by NIH Grant Nos. CA-10906 and AM-15051. The operation and continuing development of the PROPHET System is sponsored by the Chemical/Biological Information-Handling Program, Division of Research Resources, National Institutes of Health.

** A detailed description of PROPHET System features and organization is given in the accompanying paper by Castleman, et al.

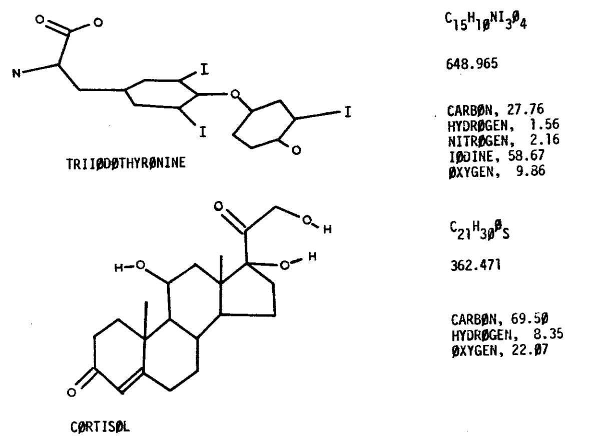


Figure 1

the bonds involving the central oxygen atom are depicted in Figure 3. All reasonable parameters which could describe the relative orientation of the two rings in the 17 thyroactive structures were stored in a table (Table I). Plots were made of various combinations of rows and columns in order to look for systematic patterns. Figure 4 is an example of a graph where no correlation between the parameters was observed. On the other hand, Figure 5 shows a discontinuity with the data falling into two widely separated groups, and Figure 6 illustrates the two overall types of molecular conformations which have been termed transoid and cisoid.

One of the most convenient features of the PROPHET

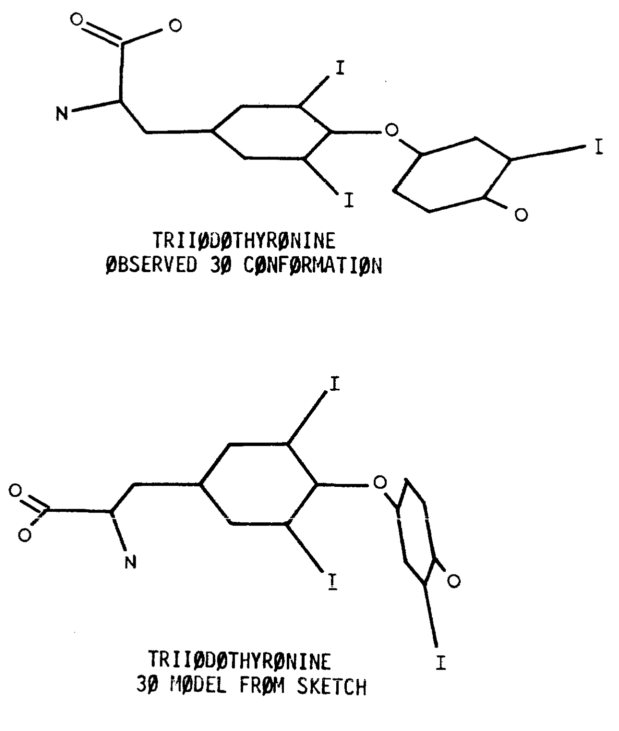


Figure 2

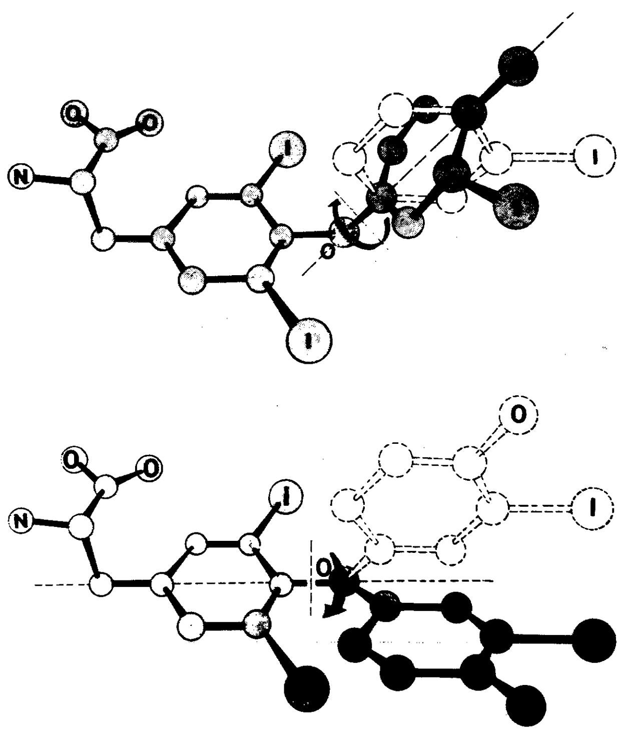


Figure 3

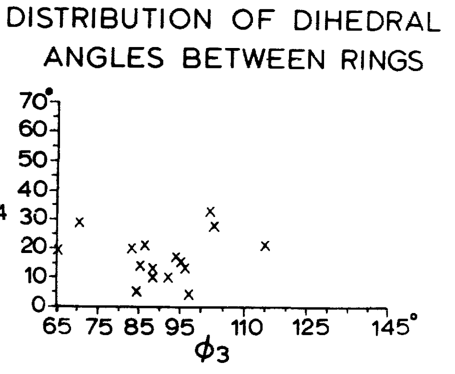


Figure 4

TABLE I—Conformational Parameters for Thyronine Compounds

NAME	Phi1	Phi2	Phi3	Phi4
T00101	86.5°	21.0°	83°	20°
T00202	-92.2	0.1	92	10
T00302	79.9	27.3	70	29
T00402	-66.8	-18.6	65	19
T00502	-96.7	10.2	96	13
TN0603	-89.8	-13.1	85	15
TN0703	86.5	13.4	85	14
TN0803	100.5	-27.2	94	17
TN0903	-88.7	-14.2	83	4
T01004	-107.9	32.9	103	28
T01105	87.6	5.2	84	5
T01206	116.3	-20.5	115	21
T01307	144.1	-67.7	143	67
T01408	89.9	-11.4	88	13
T01509			88	10
T01610	104.7	-34.4	102	33
T01711			86	21

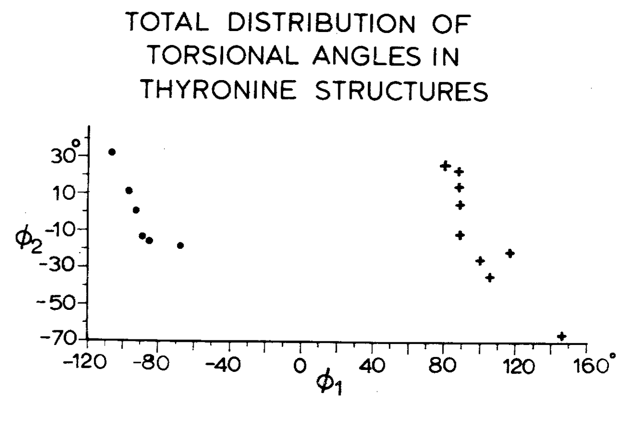
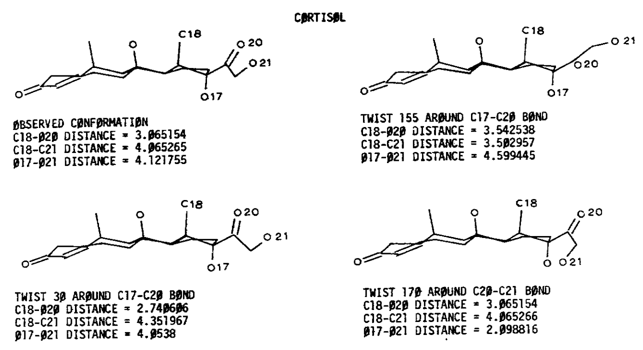


Figure 5



```
ØBSERVED CØNFØRMATIØN
C18-Ø2Ø DISTANCE = 3.Ø65154
C18-C21 DISTANCE = 4.Ø65265
Ø17-Ø21 DISTANCE = 4.121755

TWIST 155 ARØUND C17-C2Ø BØND
C18-Ø2Ø DISTANCE = 3.542538
C18-C21 DISTANCE = 3.5Ø2957
Ø17-Ø21 DISTANCE = 4.599445

TWIST 3Ø ARØUND C17-C2Ø BØND
C18-Ø2Ø DISTANCE = 2.74Ø6Ø6
C18-C21 DISTANCE = 4.351967
Ø17-Ø21 DISTANCE = 4.Ø538

TWIST 17Ø ARØUND C2Ø-C21 BØND
C18-Ø2Ø DISTANCE = 3.Ø65154
C18-C21 DISTANCE = 4.Ø65266
Ø17-Ø21 DISTANCE = 2.Ø98816
```

Figure 8

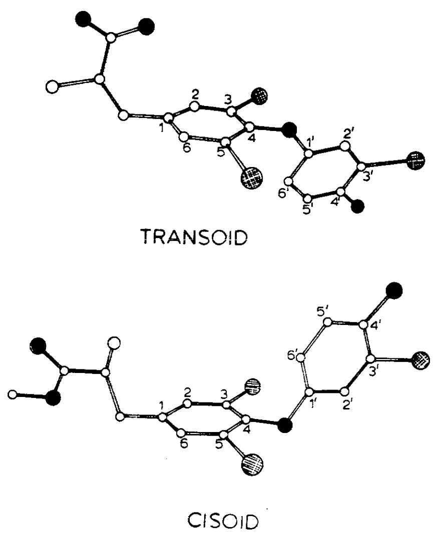


Figure 6

system is the ease with which a molecule may be viewed from different perspectives. Figure 7 shows the cortisol molecule in orientations which may be described as top, side, and end views. These pictures were produced using the TURN commands which rotate displays about the three axes.

The experimentally observed conformation of a molecule may also be modified using simple commands so that geometrical parameters of a theoretical conformation can be quickly computed. Examination of these parameters may reveal energetically unfavorable features of the theoretical conformation. For example, the orientation of the side-chain attached to C(17) in the cortisol nucleus might be thought to be flexible, but crystallographic investigation of cortisol derivatives has shown the torsional angles about the C(17)-C(20) bond to be largely invariant. As shown in Figure 8, the CHEMSET command was used to rotate the sidechain about this bond, and distances were then calculated between atoms in the nucleus and the sidechain by simply pointing to the atoms in question with the stylus. This feature of the PROPHET system makes it easy to deform a molecule by small amounts and to follow the atomic interactions at many different points.

A comparison of the structural data for cortisol and its derivatives also reveals one of the best examples of a correlation between conformational differences and biological activity which has been observed in the steroids. The effects of cortisol on carbohydrate metabolism and inflammatory re-

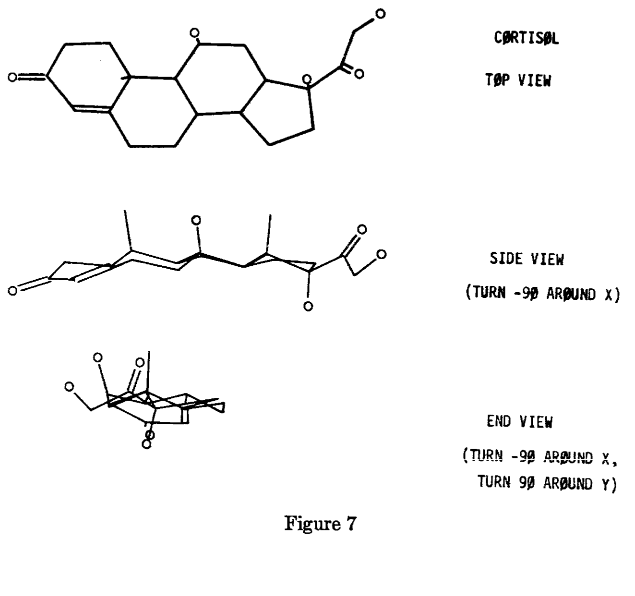


Figure 7

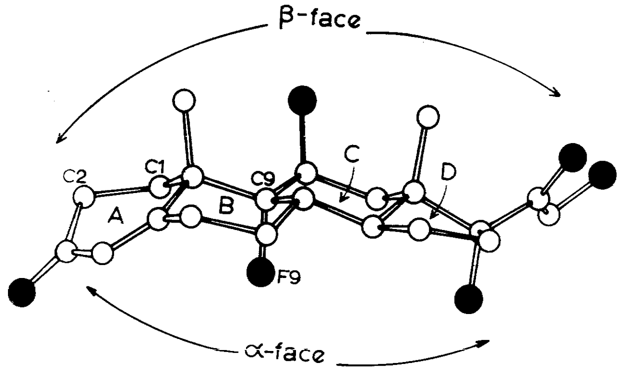


Figure 9

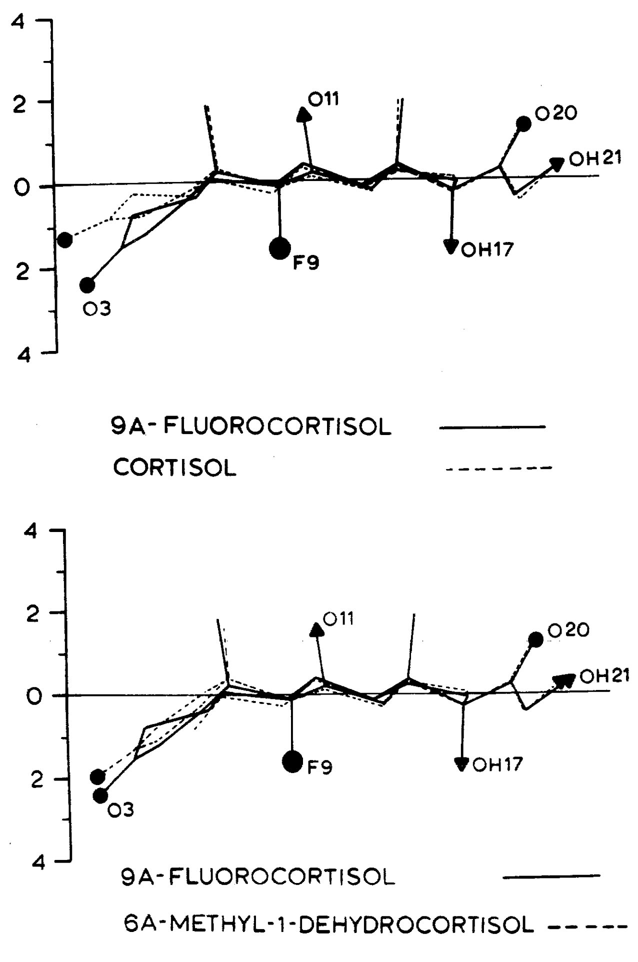
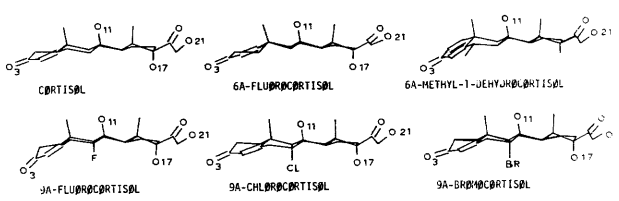


Figure 10



	INTERATØMIC DISTANCES		
STRUCTURE	Ø3-Ø11	Ø3-Ø17	Ø3-Ø21
CØRTISØL	6.846797	9.683412	12.52215
6A-FLUØRØCØRTISØL	7.Ø22464	9.7ØØ885	12.665
6A-METHYL-1-DEHYDRØCØRTISØL	6.592556	9.196394	11.9Ø5Ø9
9A-FLUØRØCØRTISØL	6.827164	9.Ø323Ø8	12.Ø9148
9A-CHLØRØCØRTISØL	6.7Ø9984	9.773992	12.4871
9A-BRØMØCØRTISØL	6.81Ø393	9.687892	12.47865

Figure 11

actions may be enhanced by a number of modifications to the steroid nucleus including dehydrogenation of atoms C(1) and C(2) and by replacement of the hydrogen at the 9-position on the bottom or α-face of the molecule with a fluorine atom (see Figure 9). A comparison of the molecular geometry of the 9α-fluorocortisol molecule with the structures of cortisol and 6α-methyl-1-dehydrocortisol suggests that the increased activity of the 9α-fluoro derivative may result, in part, from unexpected changes in the A-ring conformation. Superposition of these molecules in Figure 10 shows that the A-ring in 9α-fluorocortisol is bent underneath the plane of the molecule to a much greater extent than is the A-ring in cortisol, and in this respect, the conformation of 9α-fluorocortisol resembles the conformation of 6α-methyl-1-dehydrocortisol. From inspection of a Dreiding model of 6α-methyl-1-dehydrocortisol, it can be seen that the presence of the 1-2 double bond will force the A-ring to adopt a conformation similar to that actually observed in the crystal structure. However, the reason for the sharp bowing of the A-ring toward the α-side in 9α-fluorocortisol is not so obvious, and the bowing was not anticipated prior to the crystallographic investigation. These differences in A-ring conformation influence the distances between O(3) and other oxygen atoms in cortisol and its derivatives as shown in Figure 11. Variations of this type may be related to the ability of these molecules to bind to proteins *in vivo*.

The preceding illustrations demonstrate the utility of the PROPHET system in the search for correlations between molecular structure and biological function. The investigation of such structural-functional relationships will be greatly simplified when files of bioassay data become available within PROPHET. Greater use of PROPHET to explore mechanisms of action and to identify the effects of changes in structure will also be made in the future. Molecular model building experiments and conformational energy calculations are some of the techniques which will be used to accomplish these goals. For example, rotameric potential energy mapping procedures can be employed to make a sophisticated analysis of the interactions which occur when a molecule is deformed as was illustrated by rotation of the cortisol sidechain. These procedures produce energy contour maps which relate various molecular conformations to the energy associated with nonbonded contacts between the atoms at the ends of rotating bonds. The interactions of more than one molecule will also be investigated in order to provide models for such molecular events as the binding of a steroid hormone to its receptor protein.

Applications of the PROPHET system in molecular pharmacology—Structure-activity relationships in monoamine oxidase inhibitors*

by CARL L. JOHNSON

City University of New York and Beth Israel Medical Center
New York, New York

INTRODUCTION

Understanding the molecular bases of drug action and drug-related phenomena in man and other animal species is a primary concern of the Department of Pharmacology, Mt. Sinai School of Medicine. Toward this end, a diverse array of laboratory and theoretical investigations involving a wealth of analytical and computational tools are in progress. This paper describes one of these areas of study, namely, structure-activity relationships in inhibitors of the enzyme monoamine oxidase, and illustrates how the PROPHET System** is employed in the organization, manipulation, and analysis of research results.

BACKGROUND

Monoamine oxidase (MAO), an enzyme mainly localized in the mitochondrion of the cell, is responsible for the oxidative deamination of physiological monoamines, including norepinephrine, dopamine, and serotonin. The importance of this enzyme in controlling the levels of putative neurotransmitters has created a great deal of interest in determining the mechanism of substrate oxidation and the nature of the active site. In addition, considerable effort has been directed at understanding the mechanism of MAO inhibition by a wide variety of chemical classes, all of which exhibit a pharmacological profile including potent anti-depressant activity.

Some of the known MAO inhibitors include arylalkylhydrazines, arylhydrazides, α-methylated arylalkylamines, arylcyclopropylamines, aryloxycyclopropylamines, N-cyclopropylaryloxyethylamines, β-carbolines and arylpropargylamines. A common structural feature of these various classes of inhibitors as well as substrates is an amino group, and this group is assumed to play an essential role in complex formation at the active site. In addition, the aryl portion of the MAO inhibitors, although not an absolute requirement, is essential for the potent inhibition of the enzyme, and substitution on the aryl ring can markedly affect potency.

One approach to understanding the role of the aryl portion of these compounds and to gain insight into the general relationships between chemical structure and pharmacological activity is to carry out quantitative structure-activity studies on a series of related molecules. This approach requires, in addition to some estimate of the biological activity of the series of derivatives, a set of parameters which describe in numerical terms the structural characteristics of the compounds. The parameters commonly utilized in this type of study include empirical physical-chemical properties, such as Hammett sigma constants, partition coefficients, molar volumes, etc., and theoretical parameters, such as atomic charges calculated by quantum chemical methods. The data set, consisting of biological activities and structural parameters, is then analyzed by some form of computerized multiple regression technique.

In the present communication we report on the quantitative structure-activity relationships (QSAR) for a series of MAO inhibitors and compare the results with those reported for other classes of MAO inhibitors. In addition, by comparing the molecular structures of the various inhibitor classes, we have attempted to arrive at a general picture of the MAO inhibitor pharmacophore.

METHODS

This investigation of monoamine oxidase inhibitors makes important use of the table-handling features of the PROPHET System. By means of column headings and row names, the data type TABLE provides a natural and conceptually attractive means of data organization and an easily readable presentation. The tables presented in this communication are photocopies of the actual tables as they are displayed on the terminal. Note the utility of the feature which allows footnotes to the table entries to be stored in the form of a text associated with the individual table. The types of tabular entries generally employed in this work are fixed or floating point numbers, text, molecular structures, graphs, and sub-

* This research is supported by Grant MH-17489 from the National Institute of Mental Health. The operation and continuing development of the PROPHET System is sponsored by the Chemical/Biological Information-Handling Program, Division of Research Resources, National Institute of Health.

** A detailed description of PROPHET System features and organization is given in the accompanying paper by Castleman et al.

TABLE I

HYDRAZIDES 23R X 5C

	1 SIGMA[a]	2 LOG P[b]	3 Q(1)-PI[c]	4 M.I. (OBS)[d]	5 (D) M.I. (CALC)[e]
1. 2-FURANYL			1.1166	198.	34.2998
2. 3-THIENYL		1.6	1.0376	192.	181.0875
3. 2-PYRIDYL		.3	1.0415	191.	173.841
4. 2-THIENYL		1.61	1.1444	156.	-17.35458
5. 2-PYRAZYL			1.0587	152.	141.0822
6. 4-CL-PHENYL	.23	2.84	1.0593	134.	140.7673
7. 3-NO2-PHENYL	.71	1.85	1.065	125.	130.1763
8. 3-CL-PHENYL	.37	2.84	1.0655	123.	129.2473
9. 3-PYRIDYL		.3	1.0794	111.	103.4201
10. 4-PYRIDYL		.32	1.0335	100.	188.7056
11. 4-I-C3H7-PHENYL	-.15	3.66	1.0953	97.	73.87674
12. 3-ISOPYRROLYL			1.0871	84.	89.11293
13. 3,4-(CH3)2-PHENYL	-.24	3.25	1.0774	76.	107.1362
14. 4-OCH3-PHENYL	-.27	2.11	1.1132	63.	40.61728
15. 2-NH2-PHENYL	-.7	.9	1.1017	60.	61.98509
16. 5-THIAZOLE		.44	1.1219	38.	24.45203
17. 4-(1,2,3-TRIAZOLE)			1.1325	38.	4.75647
18. 4-NH2-PHENYL	-.66	.9	1.1025	15.	60.4986
19. 4-OH-PHENYL	-.36	1.46	1.1139	11.	39.31659
20. 2-OH-PHENYL	-.4	1.46	1.1305	0.	8.472595
21. 5-PYRIMIDYL		-.4	1.1354	0.	-.6319275
22. 4-(3,5-(CH3)2)-ISOXOA		1.2	1.2877	0.	-203.6163
23. 3-PYRIDAZYL			1.0375	0.	101.2733

[a]HAMMETT SIGMA CONSTANTS FROM MCDANIEL AND BROWN(2).

[b]OCTANOL:WATER PARTITION COEFFICIENTS FROM THE COMPILATIONS OF HANSCH AND COWORKERS (3,4).

[c]PI ELECTRON DENSITY AT THE C(1) CARBON ATOM OF THE ARYL RING FROM CNDO/2 CALCULATIONS.

[d]OBSERVED POTENCY FOR INHIBITION OF BRAIN MAO EXPRESSED AS THE MARSILID INDEX (M.I.) FROM BIEL ET AL.(1).

[e]POTENCIES CALCULATED FROM REGRESSION ANALYSIS, EQUATION (4), TABLE 2

tables, and frequent use is made of the large number of routines which are available for operating on these various data types.

The monoamine oxidase project also exercises heavily PROPHET's molecule-handling features. Molecular structures are entered by sketching in the molecule on the tablet. This provides the atom types and connectivities necessary to compute a three-dimensional model of the molecule using standard bond angles and bond lengths or user provided values. Alternative conformations of the molecule and its relative orientations in space then are examined systematically as the user tests various hypotheses about structure-function relationships. Particular use is made of System procedures for displaying a Dreiding model of the molecule with or without atom labels, in either a two dimensional projection or as stereoscopic pairs. In addition, there is frequent use of those commands which retrieve information about the molecule, such as internuclear distances, bond angles, and coordinates.

Analysis of empirical structure-activity relationships in the monoamine oxidase inhibitor class is facilitated further by the ready availability of a wide variety of structural parameters, including electronic, hydrophobic, and steric substituent constants, which have been compiled by the author and his associates and stored on the PROPHET System disk. In most SAR studies involving substitution on an aromatic ring, the user need only supply the biological data for a series of derivatives and the necessary structural parameters will be automatically retrieved and fed to the multiple regression analysis program.

RESULTS AND DISCUSSION

Structure-activity relationships of hydrazide MAO inhibitors

Arylhydrazides of the general structure

$$\text{Ar-CONHNHCH(CH}_3)_2$$

were one of the first known inhibitors of MAO. The *in vivo* potencies of a series of 23 derivatives have been reported[1] and are listed in Table I (column 4). The potencies are reported as the Marsilid Index (M.I.) and represent the relative extent, of inhibition of brain MAO, taking iproniazide (Marsilid) as the standard (M.I. = 100). Ten of the derivatives are ring-substituted phenylhydrazides and the remaining 13 contain heterocyclic aryl groups. As empirical structural parameters, we have employed Hammett sigma constants (Table I, column 1) and octanol: H_2O partition coefficients (column 2). Since Hammett constants are available only for the phenyl derivatives the empirical QSAR studies were restricted to these ten compounds. Multiple regression analysis was carried out using a program adapted for the PROPHET system. The results are shown in Table II. A significant correlation of the Marsilid Index and the Hammett constant was obtained (equation 1), electron withdrawing groups increasing activity. A much poorer correlation was obtained using the hydrophobic constant Log P (equation 2) and including this parameter along with sigma (equation 3) did not improve the correlation over that obtained with sigma alone. These results suggested that the major effect of ring substitution was an electronic one and that the partitioning behavior or hydrophobic nature of the aryl group was not a significant determinant of activity.

Since the majority of the derivatives could not be included in the empirical analysis, molecular orbital (MO) calculations were carried out using the CNDO/2 method.[5] These calculations were performed on a CDC 6600 computer. At the present stage of development of the PROPHET system calculations performed on other computers are written on magnetic tape and sent by mail to the PROPHET installation where they are then stored on disk and are available for input to the multiple regression program. We anticipate, however, that in the near future direct connections between

TABLE II

REGRESSION 5R X 6C

	1 REGRESSION EQUATION[a]	2 N	3 R	4 S	5 F	6 EQUATION NO.
1	M.I.=88.25 SIG + 83.37	10	.804	31.4	14.6	(1)
2	M.I.=12.4 LOG P + 64.5	18	.219	64.3	.81	(2)
3	M.I.=70.51 SIG + 17.13 LOG P + 44.32	10	.856	29.1	9.6	(3)
4	M.I.=-1858.1 Q(1) + 2109.0	18	.944	20.8	130.9	(4)
5	M.I.=-681.1 Q(1) + 8.23 LOG P + 820.1	18	.65	51.8	5.5	(5)

[a] ABBREVIATIONS: SIG = HAMMETT SIGMA CONSTANT; LOG P = OCTANOL:WATER PARTITION COEFFICIENT; Q(1) = PI ELECTRON DENSITY AT THE C(1) CARBON ATOM OF THE RING; N = NUMBER OF COMPOUNDS; R = MULTIPLE CORRELATION COEFFICIENT; S = STANDARD ERROR OF THE REGRESSION; F = F-TEST FOR SIGNIFICANCE OF THE REGRESSION.

the PROPHET computer and one or more large computers will be available for transmitting data in both directions.

The hydrazide data were examined for correlations of activity with a variety of MO parameters, including total and pi charges, frontier densities, dipole moments and orbital energies. A significant correlation was obtained (Table II, equation 4) with only one of the many MO parameters, namely the pi electron density at the 1-position of the aryl ring (position of attachment at the hydrazide group). Four of the 23 compounds did not fit the regression and were not included in the analysis. A fifth compound was predicted to have a large negative activity and in fact was reported to be inactive. Thus, of the 23 derivatives, only four were poorly predicted by the regression equation (predicted potencies are given in column 5 of Table I). Inclusion of both Q(1) and Log P in one equation resulted in a much poorer regression (equation 5). The results suggest that the lower the pi electron density at the 1-position of the aryl group (i.e., the less negative this atom) the greater the potency of the inhibitor. Lowering the pi density can be accomplished either with electron withdrawing groups on the ring or with heteroatoms at specific positions in the aryl ring.

Comparison of the QSAR of other MAO inhibitors

QSAR studies have recently been published[6] for a large number of MAO inhibitor classes (Table III). Although there are some differences among the various groups, and part of this could be due to different enzyme sources, experimental technique, choice of derivatives, etc., it is remarkable that many of the equations show similar parameter values. All show an apparent steric inhibition (ES) effect and most show a hydrophobic bonding effect (PI), although in some cases the coefficient is positive and in others negative. For our purposes, the electronic effect is of greatest interest and it is notable that in all cases where this type of effect is demonstrable the coefficient of this term is positive. Thus, for the examples shown in Table III and our own results of Table II, electron withdrawing groups on the aryl portion of the MAO inhibitors increase potency. Since many of the inhibitors have saturated carbon atoms between the aryl group and the amino group it is safe to assume that the electronic effects are restricted to the aromatic ring and not directed to the side chain functional groups. Since electron withdrawal increases activity it is possible that the ring interacts with an electron rich enzyme site (e.g., a tyrosine residue) through π-π stacking or charge transfer type interactions.

TABLE III

MAOI 10R X 3C

	1 REGRESSION EQUATION[a]	2 N	3 R
1. N-CYCLOPROPYL-PHENOXYETHYLAMINES	1.75SIG+.77ES+.18PI+4.00	15	.976
2. PROPARGYLAMINES	1.19SIG+.76ES+.39PI+5.55	11	.937
3. BENZYLHYDRAZINES	1.64SIG+.52ES-.55PI+5.83	8	.996
4. ALPHA-METHYL-TRYPTAMINES	1.25SIG+1.07ES-1.09PI+3.15	15	.862
5. PHENOXYCYCLOPROPYLAMINES	1.8SIG+1.25ES+5.35	6	.95
6. PHENYLCYCLOPROPYLAMINES	1.86SIG+.50ES-.75PI+5.18	10	.939
7. BETA-CARBOLINES	.72SIG+.73ES+.59PI+.36D+2.78	12	.979
8. TETRAHYDRO-BETA-CARBOLINES	.73ES+.53PI+1.0D+2.59	15	.909
9. ALPHA-METHYL-BENZYLHYDRAZINES	.93ES+.61PI+3.34	7	.876
10. ALPHA-METHYL-PHENYLETHYLHYDRAZINES	1.67ES-.68PI-.27	7	.894

[a]DATA TAKEN FROM FUJITA (6). ABBREVIATIONS: SIG = HAMMETT SIGMA CONSTANT; ES = TAFT STERIC CONSTANT; PI = OCTANOL:WATER PARTITION COEFFICIENT; D = DUMMY VARIABLE REPRESENTING PRESENCE OR ABSENCE OF N-METHYL SUBSTITUTION; N = NUMBER OF COMPOUNDS; R = MULTIPLE CORRELATION COEFFICIENT.

TABLE IV

MAOIMOLECULES 8R X 4C

	1 FORMULA <MOLECULE>	2 D(1)[a]	3 D(2)[a]	4 D(3)[a]
1. IPRONIAZIDE	$C_9H_{14}ON_3$	1.14	3.75	4.58
2. BENZYLHYDRAZINE	$C_7H_{10}N_2$	1.37	3.74	4.56
3. ALPHA-METHYL TRYPTAMINE	$C_{11}H_{15}N_2$	1.33	3.92	4.43
4. PHENOXYETHYLAMINE	$C_8H_{11}NO$	1.19	3.83	4.62
5. PARGYLINE	$C_{11}H_{13}N$	1.11	4.21	4.93
6 CIS-PHENOXY-CYCLOPROPYLAMINE	$C_9H_{12}ON$	1.44	4.19	5.03
7. TRANS-PHENYL-CYCLOPROPYLAMINE	$C_9H_{12}N$	1.49	3.86	4.48
8. MEAN+/-STD.DEV.		1.30+/-0.15	3.93+/-0.20	4.66+/-0.23

[a]INTERNUCLEAR DISTANCES (IN ANGSTROMS) IN A CONFORMATION CORRESPONDING TO THE PROPOSED MAO INHIBITOR PHARMACOPHORE. SEE TEXT FOR MEANING OF PARAMETERS.

Comparisons of the molecular structure of MAO inhibitors

The remarkable similarity in the QSAR results for the various MAO inhibitors shown in Tables II and III encouraged us to look for similar structural features in the molecules. Three dimensional models of the various MAO inhibitors were constructed in the PROPHET computer using standard bond angles and bond lengths. Then, using PROPHET's ability to manipulate molecules in the same coordinate space and to rotate single bonds, we examined the structural relationships of the amino group to the aromatic ring in the various inhibitors. All of the inhibitors show considerable rotational flexibility. However, we found that it was possible to orient the amino group of all of the inhibitors, with the exception of the propargylamines and β-carbolines, in the same relative position with respect to the aromatic ring. The results are shown in Table IV. D(1) is the distance of the amino nitrogen above the plane of the ring. D(2) and D(3) are the internuclear distances between the amino nitrogen and the C(1) and C(2) carbon atoms in the ring, respectively. Although the amino group of pargyline cannot be made to fit the same pattern as the other inhibitors, it is of interest that the acetylenic carbon atom does fit this pattern (see Table IV). However, β-carboline, being a completely planar molecule, cannot fit this pattern and has been suggested to bind to a different site on the enzyme.[7] Thus, we propose that the MAO inhibitor pharmacophore consists of an electron rich functional group (either an amino nitrogen or an acetylenic carbon) located 1.30 ± 0.15 A° above the plane of the ring and 3.93 ± 0.20 A° away from the ring. The proposed pharmacophore is consistent with the stereo-selectivity of the phenoxycyclopropylamines: cis-phenoxycyclopropylamine is more potent than the trans isomer by a factor of 10^7 and only the former isomer fits the proposed pattern. Similarly, the trans isomer of phenycyclopropylamine is more potent than the cis isomer[7] and only the former fits the pattern.

REFERENCES

1. Biel, J. H., A. Horita, and A. E. Drukker, "Psychopharmacological Agents," *Medicinal Chemistry Series*, Vol. 4-I (Gordon, ed.), Academic, New York, 1964, pp. 359-443.
2. McDaniel, D. H. and H. C. Brown, *J. Orgn. Chem.* 23, pp. 420-427, 1958.
3. Hansch, C., A. Leo, S. H. Unger, K. H. Kim, D. Nikaitani, and E. J. Lien, *J. Med. Chem.* 16, pp. 1207-1216, 1973.
4. Leo, A., C. Hansch, and D. Elkins, *Chem. Rev.* 71, pp. 525-616, 1971.
5. Pople, J. A. and G. A. Segal, *J. Chem. Phys.* 44, pp. 3289-3296, 1966.
6. Fujita, T., *J. Med. Chem.* 16, pp. 923-930, 1973.
7. Zirkle, C. L., C. Kaiser, D. H. Tedeschi, R. F. Tedeschi, and A. J. Burger, *Med. Pharm. Chem.* 5, pp. 1265-1284, 1963.

Applications of the PROPHET system in human clinical investigation*

by BERNARD J. RANSIL**

Boston City Hospital
Boston, Massachusetts

INTRODUCTION

The study of chemical-biological interrelationships reaches its most challenging levels of complexity and difficulty in human clinical investigation. Whether the objective is development of a new drug, better understanding of an old one, or further elucidation of life processes which normally or pathologically distribute, transform, and eliminate exogenous and endogenous chemical substances, there is no experimental subject more important and more complex than man. And since no experimental subject is more difficult to study systematically than man, there probably is no area of biology and medicine where computer science and technology are needed more, but exploited less, than in human clinical investigation.

This paper surveys the nature of clinical investigative research and the information-handling problems associated with it. Selected topics are illustrated by describing how the PROPHET system† has been used in the Clinical Research Center of the Harvard Medical Unit at Boston City Hospital. Among other things, the discussion attempts to show that PROPHET possesses a sufficiently large and powerful repertoire of features to make it not only an invaluable tool in clinical pharmacology but also an instructive paradigm of a protocol-oriented, research data-handling system for almost any type of clinical investigation.

CLINICAL INVESTIGATION

Human clinical investigation is both descriptive and quantitative, inductive and deductive, concerned with both individual and collective events. Not unlike statistics (defined somewhere as "the art of being precise though vague"), it possesses an applicational elusiveness arising in part perhaps from the range and actuarial nature of the phenomena it explores.

Human clinical investigation begins with observations of various types and degrees of precision and accuracy at various organizational levels, concerning the function and malfunction of the human body, made upon identifiable subjects or groups of subjects for specified periods of time under specified circumstances. The less known about a phenomenon under study, usually the more descriptive the approach. As the phenomenon, process or mechanism under study approaches less complex levels of structural organization and function (organism→organ→tissue→molecule), the greater the potential for more quantitative studies, for more rigorous experimental control, and greater investigative precision.

The activity of clinical investigation depends on data—its collection, storage, retrieval and utilization as a basis for understanding disease processes in man, and their cure, reversal or control, in a context that can range from the so-called "controlled" to "non-controlled" investigative conditions.

The "controlled" experiment attempts to study a phenomenon in terms of both a perturbed and unperturbed state, or in terms of one state compared to some defined reference state. This may be done in a number of ways, which is a topic itself in experimental design.

The "controlled experiment" is demarcated in both time and population size. Its conclusions refer to the small samples studied but are often extended (with varying degrees of validity) to populations-at-large on the basis of sampling theory, under the assumption that a true cross-section obtains. Most modern clinical investigation is conducted in this mode.

Less common nowadays is that form of clinical investigation arising from the so-called "bedside observation." As practiced by the early pioneers of clinical medicine, at a time when the practice of medicine was more personal, more descriptive, less precise, less academically oriented, this technique consisted of periodic observations of scores or hundreds of patients over time spans of months to years. It was

* Supported in part by Grant RR-76 from the General Clinical Research Centers Program of the Division of Research Resources, National Institutes of Health. The operation and continuing development of the PROPHET system is sponsored by the Chemical/Biological Information-Handling Program, Division of Research Resources National Institutes of Health.

** Address after July 1, 1974: Department of Medicine, Beth Israel Hospital, Boston, Massachusetts 02215.

† A detailed description of PROPHET system features and organization is given in the accompanying paper by Castleman et al.

inseparable from patient care and usually employed the long term qualitative and descriptive patient care data as its data base for comparison, correlation and inference. The "data bank" was one physician's files and personal experience; the "memory" his own; the primary data processing technique was "clinical correlation" using trial-and-error therapeutics with outcome (improvement, cure, no improvement, death, etc.) as a measure of therapeutic efficacy upon which the value of a drug, therapeutic regimen or accuracy of diagnosis was judged. Transmission of information was primarily by example, word of mouth and the published case history. This form of investigation—uncontrolled by modern standards, descriptive, qualitative for the most part, correlative—was open-ended as to time and population sample. Generalization to larger populations, always possessed of an element of risk even with the best controls and sampling techniques, possessed sufficient validity to produce recognizable improvements in patient diagnosis and treatment. In its own way, judging from the contribution it made to the understanding of many disease processes and the alleviation of human suffering, it worked.

CLINICAL INVESTIGATORS

Contemporary clinical investigation is performed by individuals, groups and teams possessing incredibly diverse backgrounds, training and levels of competence. They range from high school students on summer projects to multiple-degreed scientists with decades of experience in their specialties. They include individuals formally trained in medicine, the biological, physical, mathematical, engineering and social sciences.

Because most clinical investigation is rooted in the study of small representative samples of subjects or specimens, a working knowledge of statistics is mandatory. This includes a good knowledge of, and feel for, the notion of distributions (especially Gaussian, Poisson and binomial) and their central measures, the theory of sampling, error analysis and confidence limits, the basic idea of significance tests and the common types of significance tests for the most frequently encountered distributions, the notion of correlation for one or more variables, trend analysis, ranking tests, curve fitting and measures of goodness of fit, analysis of data for internal consistency and elimination of outriders.

It is probably not overstating the case to say that the acquisition of the prerequisite statistical, mathematical and computing skills is not as urgent a priority to most clinical investigators in their formal scientific training and formation as the development of expertise in the preclinical and clinical sciences. Having been exposed to the customary elementary course in statistics, usually taught by statisticians or mathematicians with cursory concern for applications, many become discouraged from further formal training by the calculational aspects and rely on picking up what they "need to know" for problem solving from colleagues and the research literature. Perusal of the clinical investigative literature reveals a tremendous variability of statistical rigor and facility not infrequently taxing credibility, despite an ever increasing demand from journal reviewers for better data reportage and analysis. It is the author's impression that many clinical investigators use biometrics not as an analytic tool to evaluate what the data actually may say about a phenomenon or a system, but as a set of required rules one follows grudgingly, mechanically, in follow-the-leader fashion to verify or rule out a working hypothesis.

DATA PROCESSING REQUIREMENTS

From the foregoing descriptions we may infer that the data processing capabilities required for clinical investigation include: collection, storage, retrieval, sorting, interfacing, transmission, analysis and diagnosis. Collection can be in several modalities—written anecdotal and descriptive data from human observers (both textual and numerical), digital or analogue data from instruments. Storage can be long or short term; if the former, the concept of a data bank arises. Data analysis can range over all forms of statistical analysis, and a variety of mathematical operations such as curve fitting and model building.

Given the nature of clinical investigation, its data processing and analysis requirements, and the general level of statistical-mathematical-computing competence of its practitioners, what characteristics might a data processing system ideally be expected to possess for clinical investigative applications?

Responses to this question can be as varied and unique as the investigators, applications and clinical research environments themselves. The author's experience in a specific clinical investigative setting has led him to identify a number of factors that must be considered. How one orders these factors as priorities to match his overall resources, constraints and goals will largely determine the system design and its hardware/software configurations. These factors include:

- Economy of initial cost, operation, maintenance and repairs, user time.
- Features important to non-specialist users: reliability—low down time, accessibility, flexibility, versatility, simplicity of operation, minimal or no programming, programmed instruction capability, interactive dialogue and batch processing modes.
- Hardware characteristics: speed (not a high priority for many applications), average down time, graphics, memory size, analogue/digital input, hard copy output.
- Nature of applications and problems to be solved: small or large (epidemiological) samples, high or low volume, one-time or repetitive solutions.
- Physical relation of labs and users.

The reader can probably supply additional criteria relevant to his particular environment and applications. In the author's setting, given the type and technical background of investigators, the nature of their problems, he has observed that the user features, i.e., accessibility, reliability, system flexibility and versatility, simplicity of operation with no programming, were given highest priority by novice users. As several investigators gained experience and confidence however, their priorities tended to shift toward wanting greater problem-solving power, table-making and data bank capability, visualization of tables and graphs with ability to produce camera-ready hard copy, thereby bypassing manual graphics production costs. As a by-product of observing this growth and developmental process, the author came to appreciate the potential of the appropriately programmed data analysis system as a programmed instruction tool for teaching statistics, graphics and mathematical analysis in the context of ongoing research projects.

It is perhaps appropriate at this point to interject a useful distinction to keep in mind between "state of the art" systems and systems designed to meet specific applicational needs when the primary goal is problem solving. The striving toward "ars gratia artis" seems as appropriate to computer science as it is to the arts and other disciplines. Yet the drive for "state of art" development as an important priority can often be inappropriate to applicational solutions where operational economy, system reliability, accessibility, stability, and ease of operation are critical priorities. Thanks to the "state of art" drive, the computer industry possesses the technical facility and know-how to produce relatively cheap, flexible, reliable data processing systems for identifiable uses. But paradoxically the "state of art" mentality is often associated with an aversion for the so-called "pedestrian" application that has an inhibitive effect on applications problem solving, obscuring the very real value, and even the "state of art" quality in its own right, of applications/problem-solving development.

All of which serves as preface to PROPHET, a biological/chemical data handling system under development by the Division of Research Resources, NIH with Bolt Beranek and Newman, Inc. and First Data Corporation. Its anatomy, physiology and functional characteristics are described elsewhere. What will concern us for the remainder of this article is the PROPHET performance in a specific clinical investigative setting, the Clinical Research Center Core Laboratory of the Harvard Medical Unit, Boston City Hospital.*

FEATURE/FUNCTION APPLICATIONS

Some idea of PROPHET's applicational relevance to human clinical investigation may be conveyed by describing how specific PROPHET features and functions are utilized in investigative problem-solving, and then how individual investigators have combined various features and functions in PROPHET procedures to achieve specific data processing and analytic capabilities tailored to their needs without recourse to programmers, keypunchers or other middlemen.

The investigative needs of the Unit for the most part require a large variety of statistical and mathematical manipulations, curve fitting, graphing and error analysis on small sample populations, usually on a demand basis. In addition, certain investigators require a capability for construction of open-ended, permanent (i.e., life of project which can be many years and several generations of investigators) data bank which may be easily corrected, extended or updated at any time, and some facility for text introduction and manipulation. Examples of typical projects and their various data processing requirements are given in Table I.

TABLE I

PROJECT TITLE	PROPHET FEATURES/FUNCTIONS
Metabolic studies of sepsis and shock in pigs and humans. Liver metabolism studies.	Data bank, statistical and mathematical analysis, graphics, profile analysis
Studies of parenteral alimentation in humans and dogs. Substrates and control mechanism of protein loss in trauma	Data bank, systems control and accounting procedures, statistical and mathematical analysis, text, graphics, profile analysis
Potassium metabolism and Na-K-ATPase activity in the isolated perfused kidney.	Data bank, statistical and mathematical analysis, graphics
Mineralocorticoid metabolism and relation to O_2 consumption.	Make table procedures, statistical and mathematical analysis, graphics
Insulin clearance in renal insufficiency.	Multiple-exponential fit and graphics from tables
Effect of exercise or systolic time intervals.	Trend analyses and graphics
Epidemiological study of cardiopulmonary resuscitation at BCH.	Trend analysis, graphics, statistical significance
Relation of amino acid levels to lipid, carbohydrate and protein metabolism in normal and diabetic, pregnant and nonpregnant subjects.	Pen-tablet input of absorption curves for amino acid determinations, graphical and error analyses from tables by column and row
Charge density basis of chemical reactivity and biological activity.	Three dimensional plotting, determination of non-integer functional dependencies using "Derived" feature of table and linear curve fit commands, correlations

PROPHET meets such needs first and foremost by the "MAKE TABLE" design feature whereby data are entered via keyboard as a permanently stored table which may then be addressed by numbered row or column to perform a variety of statistical, mathematical and graphical operations, using simple English word commands of specified syntax in a dialogue mode. Appropriate commands and dialogue yield graphs (with full control over formatting), straight-line fits to curves, measures of goodness-of-fit to plotted curves, polynomial fits of any degree, correlation studies and so on. The table feature (foreshadowed a decade ago by the stored arrays of the National Bureau of Standards' OMNITAB System*) has many practical and significant uses and consequences. It forces the investigator to organize himself and his data in a more systematic manner than is often the case, setting the stage for better analysis and therefore better

* The Harvard Medical Unit, Boston City Hospital is in process of relocating to the Beth Israel Hospital, Boston, Massachusetts.

* OMNITAB: A Computer Program for Statistical and Numerical Analysis. NBS Handbook 101, U.S. Gov. Printing Office, 1966.

basis for inference. Once the table is made, a camera-ready copy for slides or prints can be made. Statistical or mathematical analysis is accomplished by simply addressing rows or columns, using the appropriate commands (comprising English words and specific syntax) e.g., "Fit line to col 1 vs col 4 of Tablename." (The PROPHET command language currently accepts only column addressing for statistical and mathematical operations, but several investigators have written row procedures as well.)

Functional relationships of any kind may be computed from any column or combination of columns by the "DERIVED" command in the "MAKE TABLE" dialogue. This feature alone is a significant time saver where large numbers of cases are at stake, yielding derivations that require no number-by-number checking.

As useful as all this is to the clinical investigator in obtaining derived clinical parameters involving such things as drug dosages per unit area, dose response curves, concentrations of labeled drugs corrected for background, and cardiopulmonary parameters from catheterization data, even greater power is realizable in such applications as linear and non-linear regression. Regression is a term statisticians use to refer to a least squares fit of an independent variable against a dependent variable. Most often only one independent variable and a linear dependence (first degree polynomial) is assumed—whence the regrettable but ubiquitous term "linear regression." Linear regression studies are quickly performed on PROPHET by the command "Fit line to col 3 vs col 24 of Tablename." The output consists of the best straight line fit, correlation coefficient, standard deviation from regression and several other "goodness of fit" criteria, produced as hard copy in a few seconds.

Appropriate combination of the "Derive column" and "Fit line" features of PROPHET further enable the investigator to free himself from his dependency on "linear dependence" and experiment at will, in a matter of seconds to minutes, with other functional dependencies simply by making columns of any integral or non-integral power of the independent variable, invoking the command "FIT LINE . . ." to the appropriate columns and comparing the "goodness of fit" data (which include the correlation coefficient, standard deviation from regression, standard deviations of the slope and intercept, and significance of the slope) for the best fit within the precision limits of the data being analyzed.

What is done for the single variable linear and non-linear functional analyses can be done for multiple variable dependencies—the so-called multiple regression. Here again most clinical investigators who use the technique confine their analytic efforts to linear dependencies, unless other relationships are known to hold. Although multiple regression is available as a Public Procedure only for linear and exponential dependency, PROPHET's "Derive column" feature combined with the multiple regression feature allow the investigator to test any non-linear functional dependency by a series of simple commands or procedures, opening up regions of functional relationships hitherto rarely explored because of their inaccessibility to the average clinical investigator.

The value of the graph feature resides in the inherent superiority of picture over text for instant and unambiguous identification (to the trained eye) and communication of functional relationships. Thanks to its design, the PROPHET "MAKE GRAPH" feature gives the investigator full control over graph formatting, scaling, titling, curve identification and the like. Because it is quick and easy to use (from data previously stored in tables or arrays) investigators freely use it as it ought to be used—as the first step in determining functional relationships. In the majority of applications visual inspection suggests the most probable functional relationship(s) to be explored, which may then be carried out by making the necessary table alterations and functional dependency calculations. Where several functional dependencies are possible, the "goodness of fit" data may be used as a basis for ruling out one or more possibilities. If all fits yield comparable "goodness of fit" criteria, other criteria, based on additional knowledge of the system under study, if available, may be invoked; failing that, Occam's Razor (or Law of Parsimony)* is employed.

The "MAKE TEXT" feature finds a variety of uses in footnoting, annotating graphs, and report writing interspersed with graphs and tables.

The pen and tablet entry feature speeds up data corrections or alterations in tables. It also enables the investigator to move graph and table displays to any part of the screen, scaling them at will. Combining these capabilities he may display four to six graphs (depending on size and complexity) simultaneously on one page of copy for sequential analysis studies. The Public Procedure "% Points" allows data entry from a graph or drawing placed upon the tablet, enabling the investigator, via appropriate procedures, to perform curve fits and "area under curve" calculations directly from analogue recorder output tracings.

In all the foregoing applications the hard copier receives steady employment to produce both temporary and permanent copy of tables, graphs, arrays and text displayed on the screen for reference, communication and publication. To the clinical investigator it is an invaluable design feature.

Because clinical investigative data at this center is largely of the small sample type, special attention has been paid to the concept of "sample processing." Built into the "MAKE SAMPLE" software is the ability to construct a sample from any segment of a table or array, test it for the type of distribution to which it belongs (currently it tests for Gaussian, but will ultimately test for Poisson and binomial as well), and compute central measures, moments (skewness, kurtosis), median, minimum and maximum. Further, the graphics feature encourages the construction of histograms for quick, visual evaluation of population sample distribution characteristics. And the text capability allows carrying along pertinent descriptive explanations and footnotes as the need arises.

* "Entia praeter necessitatem non multiplicanda" which, freely rendered in this context is "If offered a choice between two hypotheses or equations, choose the simpler." Cf. Mathematical Approach to Physiological Problems by Douglas S. Riggs, Williams and Wilkins, 1963, p. 51.

Why the concern over "sample processing"? Because medical investigators, for historical reasons and the practical difficulties related to extensive calculational efforts (prior to the invention of low cost high speed computing devices) usually assume their data to be normally (Gaussian) distributed; and the mean and standard deviation the only sample measures that possess clinical meaning. Specific exceptions to this generalization are recognized, such as samples of cell counts (which obey Poisson statistics), drug titers (usually requiring log-normal transformations and the use of geometric means) and data that are binomially distributed. Generally speaking however, in all situations not previously established to be other than Gaussian, and in studies of new phenomena, most investigators will assume a normal distribution because few possess the innate curiosity and/or statistical know-how to test for type of distribution. The PROPHET System significantly changes all this by converting a seemingly hopelessly difficult (to the non-statistician) and time consuming task to a simple matter of typing a few words of command and answering "Yes" or "No" as required.

The revolution this obvious and simple design feature portends is suggested by a recent experience with one of the projects passing through the Core Lab data analysis facilities. One clinical investigator became interested in reviewing the status of the white blood cell (WBC) count as a clinical indicator for infectious disease (ID). He arduously collected samples of hundreds of patients in each of several categories: ambulatory patients with and without ID, hospitalized patients with and without ID, and patients with various types of ID. Because "unless you look for something, you'll overlook it," he was advised to examine the basic distribution characteristics (beginning with histograms) of several typical samples, though he was strongly inclined by force of habit to accept the work of many investigators before him and assume a normal distribution. To his surprise, he found that all samples possessed skewed normal distributions, calling for the use of geometric means and a non-symmetric confidence range rather than the familiar symmetrical confidence limits. With further prodding he saw that the degrees of skewness and the confidence range roughly correlated with the clinical condition, a new diagnostic concept in this setting. By now excited by the possibilities, he went back to the early studies of WBC counts in normal and sick populations. Some of the early German literature contained tables of raw WBC counts which, on recalculation, yielded results compatible with his own. The entire experience significantly improved his attitude toward statistics and computing systems.

Another example of the relevance of the sample feature to routine clinical investigation occurs in applications of Student's 't'-test, invented by Fisher to provide a criterion for deciding whether two sample distributions belong to the same population sample, or indeed are significantly different from one another. One of the basic requirements of the 't'-test is that the distributions be comparable and possess a mean and standard deviation. To fulfill this requirement rigorously, the full distribution characterization (type, mean, variance, sample number, skewness and kurtosis) should be known. If Poisson or binomial, the appropriate central measures must be calculated. If the variances are significantly different (determined by the F-ratio test) a correction is applied, a refinement that becomes important at borderline levels of significance. Finally if the two distributions possess significantly different degrees of skewness or kurtosis, they most likely reflect two significantly different sampling situations, which in itself may have significant clinical or investigative ramifications.

In short, what the PROPHET System sample feature provides for the medical investigator is the statistical expertise and rigor of a Fisher and a Snedecor, with none of the calculational tedium, providing both a learning experience and tool for exhaustive analysis of samples and the phenomena they represent. The investigator may ask more than elementary questions about the functional characteristics and relationships pertinent to his data with full confidence he will be able to get answers almost as readily as he can formulate the questions without calculational tedium, thereby freeing him to think about and test his data in many ways which could result in a stronger basis for inference and prediction.

This catalogue of PROPHET applications to clinical investigation by feature and function is by no means complete or all-inclusive, but it does give some idea of what is possible and provides a framework for describing how these features may be used to solve problems as they arise in the course of a research project, and how they may be combined in procedures, using PL/PROPHET programming language, to produce custom-made data processing systems for specific clinical investigative applications by the investigator himself.

PROJECT APPLICATIONS

At this center, clinical investigative applications of PROPHET fall into two categories—those which accept and utilize its basic design features, utilizing its command language and syntax, and those having additional requirements that are met by recourse to the definition and writing of procedures in PL/PROPHET.

Two ongoing projects involve the construction of a data bank, for which the investigators wrote procedures enabling them to load, delete and alter data in a single command, multiple entry mode, not possible with the usual table commands.

One such study, originating in the Harvard Surgical Unit at Boston City Hospital, involves the collection of some six dozen parameters for a well characterized patient population. Study specifications required the construction of an open-ended data bank allowing indefinite addition of patient parameters with indexing and accounting capabilities, the storing of each patient's identification, clinical and research data and the calculation of such output parameters as: cumulative weight loss, body surface area, sodium and potassium balance, total nitrogen balance and total nitrogen per unit body area, change in urinary nitrogen, creatinine clearance, mean blood pressure, cumulative nitrogen loss, and insulin/glucagon ratio.

The sequence of procedures written in PL/PROPHET to accomplish this, is itself monitored by a system of accounting tables and procedures. A procedure called TRAFFIC, for example, records the name of the table operated on, the name of the procedure called, the date and investigators' initials, the volume of information recorded by type, and the speed of program execution. Two procedures, STATUS and COLCOUNT tell the investigator what kind of data is present in a specified table or set of tables without giving the values. READOUT allows for a quicker display of non-empty values in a series of tables than is allowed by the usual PROPHET table commands.

Output procedures include PRINTNOTES (which prints out a specific patient's study notes), PRINTPTAB (which does the same thing for patient research data tables), RANGESCAN (which prints out the name of any patient specified and the number of values below or above normal of a given clinical laboratory test), GRAPHS (which makes up serial graphs having the same curves and axes from different patient tables) and CODESORT (which searches through a specified set of patient studies and locates a specified set of measurements associated with a specified event code).

In constructing a data processing system to meet the project's unique needs, the investigator utilized the command language and procedures writing capabilities, the text, table and graph features and many of the statistical and mathematical computation functions. Because the design involved many patients, many input and output parameters, with several investigators operating the terminal, additional PROPHET procedures to monitor, oversee and perform accounting so as to stay in control of, and obtain some idea of the magnitude of, the data processing activity seemed appropriate and has proven its worth to the group.

The above project is a typical example of a frequently occurring class of clinical investigative problem—a multi-parameter study designed to permit correlation of a large group of factors with some relevant aspect or facet of the patient's response to a drug, or his overall clinical condition or course, with hopefully, the emergence of some predictive capability as to outcome.

Typical approaches to such multi-parameter correlations with clinical condition are multiple regression; factor, cluster and discriminant analysis; and profile construction coupled with pattern recognition. The latter technique has been utilized by two investigators to follow a group of subjects' clinical courses over a period of time. One such study took 15 clinical observables in a specific order, collected for five subjects over a 24 hour span at three hour periods, divided each set of values by the control values and arrived at a series of curves (patterns, signals) that showed the departure of the complete sequence from control (straight line with zero slope, unit intercept) as a function of time, which in this case, correlated with a deterioration in clinical condition (sepsis in pigs). These diagrams clearly identified a group of parameters that on the average remained constant, another that decreased and a third that increased with respect to control as a function of time. It remained for the investigator to take each group and rationalize each correlation in terms of physiologic mechanisms and metabolic pathways. To the best of the author's knowledge, the investigator is still so occupied.

A third frequently occurring application of interest to the clinical investigator is a quick method for computing areas under curves and calculating best-fit functions or non-linear curves. One investigator utilized PROPHET's pentablet input device and the "% Points" entry procedure written by a Bolt Beranek & Newman staff member to determine the area under a Gaussian curve in order to compute amino acid concentration from a strip-chart recording. To utilize this procedure, the analyst simply lays the recording over the tablet and calls the procedure. Then, responding to prompting dialogue he identifies the acid, run number and date, inputs axes reference points, supplies appropriate background levels, enters each point on the tracing by a slight pen pressure on the tablet, proceeding in this manner with each curve until the last. On completion of the tracing, a table identifying each acid peak with its area, half width, maximum point, and associated concentration, together with certain measures of experimental error is printed out. While the PROPHET system is, in principle, ideal for this application, attempts to use the procedure when response time is slow are less efficient than an alternative (but not equivalent) programmable calculator routine.

USER ACCEPTANCE

This account would not be complete without some description of user use, acceptance and impact. By and large as pointed out earlier the average clinical investigator is poorly prepared for mathematical and statistical analysis, and for any technology not directly related to and used frequently in patient care/research applications. Most come to research activity from medical school, internship or residency, steeped in the anatomy, physiology and biochemistry of their special interest career, but somewhat timorous about such basic tools of their trade as bioanalytical instruments, monitoring devices and computers. They recognize the need to "know this stuff," but emulating the example of many senior staff, often try to relegate it to a technician or a knowledgeable medical student spending elective time on a research project.

Accordingly, most users of PROPHET at this location approach it innocent for the most part of previous computing experience, not a little anxious about such "technical" involvement and the threat of competition it represents to "time that should be spent on the wards" or "at the bench." The introduction therefore is one-to-one, personal, systematic, step-by-step, utilizing where possible data from the investigator's ongoing research project. One hour of such instruction usually finds the novice investigator irrevocably hooked. Thereafter most proceed on their own, using the PROPHET manual as a kind of programmed instruction device, directing questions not answered in the manual to other users or to the software systems representatives at Bolt Beranek &

Newman. Most users master enough of PROPHET in this manner to make effective use of it in their research activities without recourse to writing procedures or programs.

CONCLUSION

In summary, the PROPHET System, though still under development, has proved an invaluable research tool in this clinical investigative environment for the following reasons:

(1) With proper orientation it is readily mastered and efficiently utilized by investigators with little or no previous computer experience.
(2) It enables every investigator to apply rigorous statistics and mathematical analysis with a few simple commands, thereby stimulating his development in statistical/mathematical analysis. In this respect it functions as an efficient programmed instruction tool for teaching better biometric design and analysis.
(3) While it provides a relatively wide range of biometrical techniques without programming effort to the user, it also accommodates the user who also can program, allowing him to program and incorporate his own procedures into the basic system.
(4) Its capacity for data bank construction and table sharing has greatly facilitated long-term analysis of data and the communication of data among groups of investigators.
(5) It has been singularly responsive to user problem-solving needs.

ACKNOWLEDGMENT

The author gratefully acknowledges the contributions to this manuscript of the PROPHET users at this center, especially Dr. Robert Auty and Dan Keatinge, arising from the descriptions and documentation of their PROPHET procedures.

A simple distributed systems approach to manufacturing information systems

by LELAND R. KNEPPELT

Industrial Nucleonics Corporation
Columbus, Ohio

INTRODUCTION

The number of application packages for the production and inventory control function has steadily increased over the past ten years. The successful installations have been limited in number but the documented resultant savings are major.[1,2] These results combined with the ever present need to increase productivity and offer better customer service have furnished much of the impetus to automate effective manufacturing control techniques.

The American Production and Inventory Control Society (APICS) has generated a modern-day crusade in an attempt to focus attention on just one of the major techniques, material requirements planning. IBM through its PICS (Production Information and Control System) application packages and the recently released COPICS (Communications Oriented Production Information and Control System) manuals has generated interest among both the manufacturing and data processing personnel. With all this interest and education material, the manufacturing users are demanding the data processing systems and support for obtaining the promised cost savings.

The Industrial Nucleonics Corporation is a Columbus, Ohio based firm specializing in process automation systems. Major industry support includes pulp and paper, rubber, plastics, tobacco, and metals. The systems installed involve sensors, control devices, data collection equipment, and computers. The manufacturing function produces a large variety of configurations of fairly complex products. Product structures include both metal fabrications and electronic assemblies. This environment requires effective production and inventory control to maintain customer service and delivery objectives.

PROBLEM

Generally manufacturing systems have been plagued with a multitude of problems. In most companies the data processing installation is controlled by the financial organization within the company. Thus, any development and subsequent scheduling of manufacturing systems usually has the lower priority than the more established accounting systems. This is even the case when substantial paybacks can be realized through effective utilization of manufacturing control systems.

The manufacturing area is also at fault through their many failures to effectively utilize previously automated manufacturing control systems. They often lack the necessary formal system methods, solid objectives, and goals required to maintain control. Inaccurate inventory records, inaccurate bills of material, and unrealistic production schedules prevent any system from assisting in the control of their environment. When business conditions warrant increased production, the informal systems comprised of hot lists and expediting are used in an attempt to maintain control. The automated manufacturing systems are hard pressed to maintain a log of what has happened.

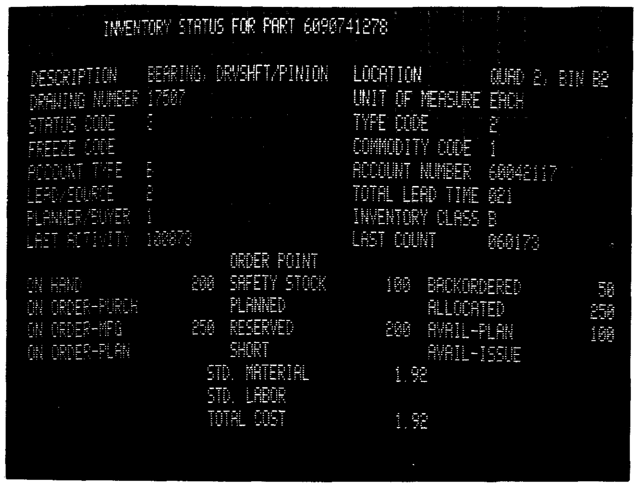
The approach of systems development to counter these problems is the use of on-line terminals to insure that the manufacturing control systems can provide accurate and timely information for production and inventory control decisions. This results in the requirement for additional hardware, software, and systems development to handle the communications environment. However, many of the problems remain, such as, the priority of scheduling and installation responsibility within another functional department.

MINI/MAXI

An alternative approach is the use of a minicomputer under the responsibility of the production and inventory control department combined with the utilization of a large scale batch computer on a "as required" basis. This concept, referred to as Mini/Maxi, provides a simple distributed systems approach to the implementation of production and inventory control applications.

The minicomputer operates in an on-line and real-time systems environment providing current status and control information. The maxi provides batch processing which is used to accomplish the more extensive applications of long range planning and manufacturing standards maintenance.

The approach places a total system under the control of the operating department with all the associated responsibilities of the data base integrity. User involvement was one



```
          INVENTORY STATUS FOR PART 6090741278

DESCRIPTION    BEARING, DRVSHFT/PINION   LOCATION         QUAD 2, BIN B2
DRAWING NUMBER 17507                     UNIT OF MEASURE EACH
STATUS CODE    3                         TYPE CODE        2
FREEZE CODE                              COMMODITY CODE   1
ACCOUNT TYPE   B                         ACCOUNT NUMBER   60042117
LEAD/SOURCE    2                         TOTAL LEAD TIME  021
PLANNER/BUYER  1                         INVENTORY CLASS  B
LAST ACTIVITY  100873                    LAST COUNT       060173
                       ORDER POINT
ON HAND             200 SAFETY STOCK      100  BACKORDERED          50
ON ORDER-PURCH          PLANNED                ALLOCATED           250
ON ORDER-MFG        250 RESERVED          200  AVAIL-PLAN          100
ON ORDER-PLAN           SHORT                  AVAIL-ISSUE
                      STD. MATERIAL       1.92
                      STD. LABOR
                      TOTAL COST          1.92
```

Figure 1

of the key ingredients for the successful implementation. This was accomplished through the establishing of ambitious but realistic goals which have to be met to justify the system. Production personnel are not new to this technique since they have had similar experiences with the purchase of new production equipment, such as, numerical control equipment. This does, however, vary from the normal data processing environment since equipment justification is usually accomplished in the systems development group or data processing department.

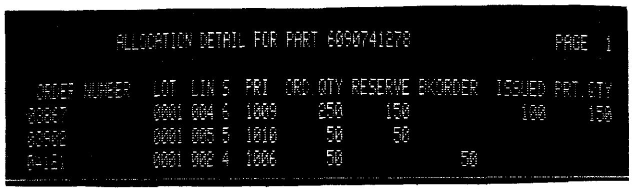
HARDWARE

The Digital Equipment Corporation's PDP 11/40 operating under the RSTS time sharing operating system was chosen for the mini hardware. The application programs were written using the BASIC PLUS language. The configuration includes disk, magnetic tape, and a line printer. The primary terminals used are the Digital Equipment Corporation VT05 Alphanumeric Display Terminal (CRT) with the RT90 Data Collection terminal available for shop floor data collection.

The maxi hardware is an IBM 370/145 operating under OS/VS1. The programs are implemented using both COBOL and assembler.

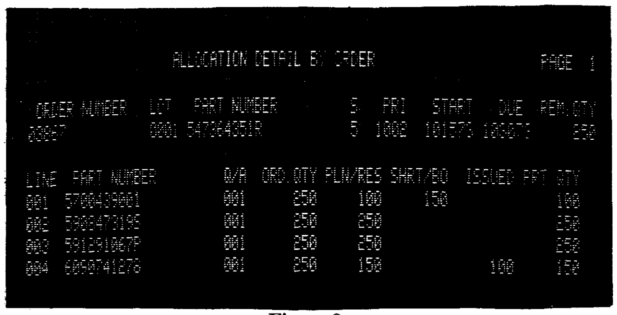
APPLICATIONS

The previously stated use of the mini/maxi approach was to implement current status and control information systems in the on-line environment of the minicomputer. The maxi



```
          ALLOCATION DETAIL FOR PART 6090741278                PAGE  1

 ORDER NUMBER   LOT  LIN S  PRI  ORD.QTY RESERVE BKORDER  ISSUED PRT.QTY
 03067          0001 004 6  1009    250     150              100     150
 03902          0001 005 5  1010     50      50
 04121          0001 002 4  1006     50              50
```

Figure 2



```
              ALLOCATION DETAIL BY ORDER                      PAGE  1

 ORDER NUMBER   LOT  PART NUMBER        S  PRI   START  DUE    REM.QTY
 03067          0001 547364351R         5  1002  101573 100873     250

LINE  PART NUMBER          Q/A  ORD.QTY PLN/RES SHRT/BO  ISSUED PRT QTY
001   5700439001           001     250     100     150              100
002   5300473195           001     250     250                      250
003   5912910G7P           001     250     250                      250
004   6090741278           001     250     150              100     150
```

Figure 3

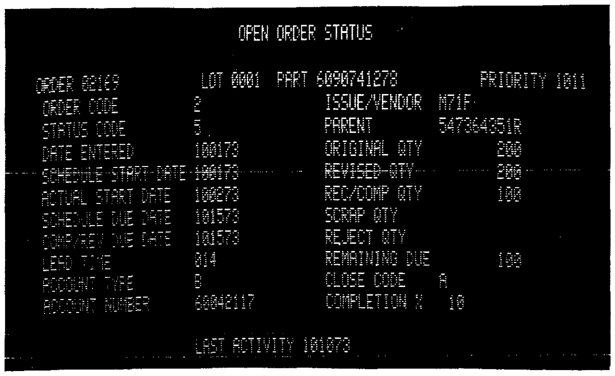
computer is then only utilized in a batch environment for the long range planning and manufacturing standards maintenance systems which require extensive magnetic file capacity, complex file management techniques, and extensive internal processing capability. The successful implementation required a comprehensive understanding of the various manufacturing control techniques to arrive at the proper allocation of application functions and data base responsibility. The next few sections provide a brief review of the key manufacturing control systems and the resultant distribution of functions between the two processing units.

FORECASTING

A forecasting system was necessary for the establishment of a master production schedule in a time frame to allow adequate production and inventory planning. It also provides assistance in the maintenance of order point and safety stock quantities for items controlled via this inventory replenishment technique.

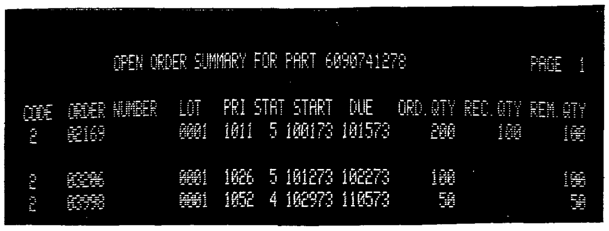
The automated portion of this system consists of only two major tasks; projection and tracking. The projection task provides a time phased demand projection using either an intrinsic method (based on past demand) or a qualitative method (projection accomplished manually outside the system and input). The tracking task is merely the compare of actual demand versus the projected demand which provides a signal when the deviations beyond established units may require a forecast adjustment.

The forecasting system is periodic batch run which can best be accomplished on the maxicomputer. The capture of



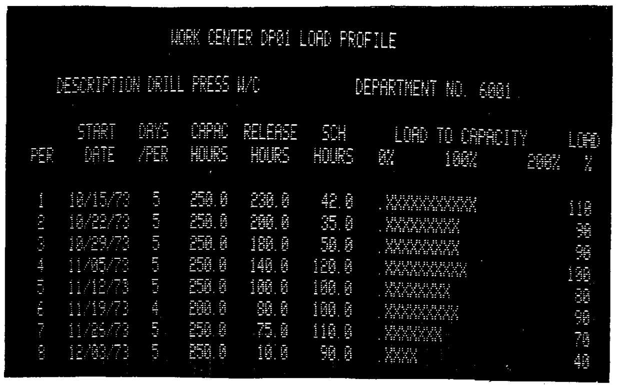
```
                    OPEN ORDER STATUS

ORDER 02169          LOT 0001  PART 6090741278          PRIORITY 1011
 ORDER CODE          2                ISSUE/VENDOR  M71F
 STATUS CODE         5                PARENT        547364351R
 DATE ENTERED        100173           ORIGINAL QTY        200
 SCHEDULE START DATE 100173           REVISED QTY         200
 ACTUAL START DATE   100273           REC/COMP QTY        100
 SCHEDULE DUE DATE   101573           SCRAP QTY
 COMP/REV DUE DATE   101573           REJECT QTY
 LEAD TIME           014              REMAINING DUE       100
 ACCOUNT TYPE        B                CLOSE CODE    A
 ACCOUNT NUMBER      60042117         COMPLETION %   10

                 LAST ACTIVITY 101073
```

Figure 4



```
               OPEN ORDER SUMMARY FOR PART 6090741278                PAGE  1

CODE  ORDER NUMBER    LOT   PRI STAT START  DUE    ORD.QTY REC.QTY REM.QTY
 2    02169           0001  1011  5 100173 101573      200     100     100

 2    03296           0001  1026  5 101273 102273      100             100
 2    03998           0001  1052  4 102973 110573       50              50
```

Figure 5



```
                     WORK CENTER DP01 LOAD PROFILE

    DESCRIPTION DRILL PRESS W/C             DEPARTMENT NO. 6001

       START    DAYS   CAPAC  RELEASE   SCH      LOAD TO CAPACITY        LOAD
PER    DATE     /PER   HOURS   HOURS   HOURS   0%       100%      200%    %

 1   10/15/73    5     250.0   230.0    42.0  .XXXXXXXXXXX                110
 2   10/22/73    5     250.0   200.0    35.0  .XXXXXXXXX                   90
 3   10/29/73    5     250.0   180.0    50.0  .XXXXXXXXX                   90
 4   11/05/73    5     250.0   140.0   120.0  .XXXXXXXXXX                 100
 5   11/12/73    5     250.0   100.0   100.0  .XXXXXXXX                    80
 6   11/19/73    4     200.0    80.0   100.0  .XXXXXXXXX                   90
 7   11/26/73    5     250.0    75.0   110.0  .XXXXXXX                     70
 8   12/03/73    5     250.0    10.0    90.0  .XXXX                        40
```

Figure 7

the actual demand figures require a constant update resulting from demand or issues from the physical stock locations. This task was therefore allocated to the minicomputer system. The transfer of actual demand figures is then made from the minicomputer to the maxicomputer for subsequent execution of the forecasting runs.

INVENTORY CONTROL

The maintenance of item status both on-hand and on-order is essential for effective inventory planning. Normally this was accomplished via manual recording of the stockroom transactions on an inventory ledger card or via batch input of the transactions for the production of a periodic stock status report. Record accuracy was marginal both from the lack of formal procedures and the timeliness of recording.

The minicomputer is used to accomplish inventory status maintenance in a real-time environment. Stockroom transactions are entered via the terminals with the associated data elements updated. Figure 1 is an example of the item status display available upon request.

The maintenance of current inventory balances is the heart of the inventory control system but additional status information is required to maintain control and support the planning systems. The minicomputer system is therefore expanded to maintain the status of reservations against an item, purchase orders, production orders, operations status of production orders, and work center load status.

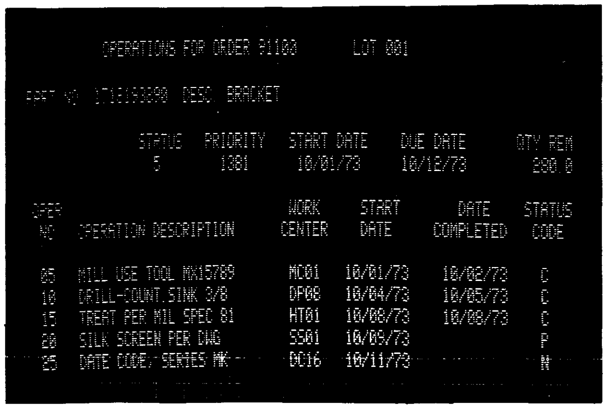
Figure 2 is an example of the reservations (or allocations) against an item. This feature provides the information regarding what production orders have reserved this item for subsequent production. Figure 3 is an example of a related display illustrating what items (components) are required to accomplish a production order and their availability. This feature eliminates the need for physical material staging by allowing a check of component availability before the actual release of the production order.

Figure 4 is an example of the production or purchase order status display. The status of replenishment orders for an item is essential for the effective use of material requirements planning. The system maintains their status through initial planning to completion. In addition to the order status display, a display of all order for particular item (Figure 5) is available. This display and function provides the time-phased on-order balance information.

The control of production orders required the minicomputer system be capable of providing shop floor control information. The system provides the maintenance capability for operation steps necessary to complete the production of an item. Figure 6 is an example of the operation status for a production order. Status is updated via operation completion information entered through the terminals. This information also provides the base for maintaining work center load status.

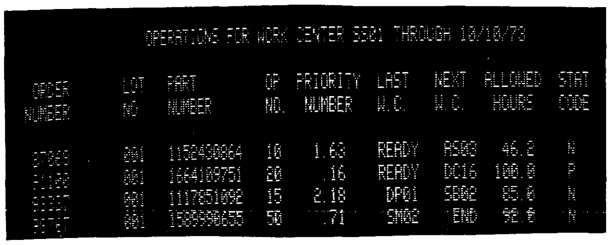
Figure 7 is an example of a work center load profile illustrating the load verses capacity for an eight time periods. The detail make-up of the load can also be accessed via the terminal. Figure 8 is an example of the detail operations within a work center.

The previous paragraphs have illustrated some of the typical displays generated via the minicomputer inventory control system. In addition it provides the transaction processing and the production of hard-copy reports upon request or on an exception basis. The hard-copy reporting includes reorder lists, cycle counting reporting, dispatch lists, job packets, material requisitions, and inventory pick lists.



```
          OPERATIONS FOR ORDER 31100       LOT 001

PART NO  1718193890  DESC  BRACKET

              STATUS  PRIORITY  START DATE   DUE DATE      QTY REM
                5       1381     10/01/73    10/12/73       200.0

OPER                               WORK    START      DATE     STATUS
 NO  OPERATION DESCRIPTION        CENTER   DATE    COMPLETED    CODE

 05  MILL USE TOOL MX15789         MC01  10/01/73   10/02/73     C
 10  DRILL-COUNT.SINK 3/8          DP08  10/04/73   10/05/73     C
 15  TREAT PER MIL SPEC 81         HT01  10/08/73   10/08/73     C
 20  SILK SCREEN PER DWG           SS01  10/09/73                P
 25  DATE CODE/ SERIES MK          DC16  10/11/73                N
```

Figure 6



```
              OPERATIONS FOR WORK CENTER SS01 THROUGH 10/10/73

ORDER       LOT  PART        OP  PRIORITY  LAST   NEXT  ALLOWED  STAT
NUMBER      NO   NUMBER      NO.  NUMBER   W.C.   W.C.   HOURS   CODE

07869       001  1152430864  10    1.63   READY   AS03    46.2    N
31100       001  1664100751  20     .16   READY   DC16   100.0    P
[illegible] 001  1117651092  15    2.18    DP01   SB02    85.0    N
[illegible] 001  1508990655  50     .71    SM02   END     92.0    N
```

Figure 8

BILL OF MATERIAL

The bill of material system provides for the storage and retrieval of the various product structures of items produced by the manufacturing function. The majority of this information is not required on a fast access basis, but is essential to production control for both material requirements planning and item identification. The automation and data base portion of this system is accomplished on the maxicomputer.

The data available consists of the bill of material (or parts list) and standard manufacturing routing for each item which can be produced. Reporting for the system includes the various explosions, implosions, standard routings, where-used list (work center and item), and engineering notes. Request for the reports as well as data base maintenance is captured on-line through the minicomputer. They are then transmitted to the maxicomputer for subsequent processing. The actual reporting can be output on the maxicomputer or transmitted for printing on the minicomputer.

The bill of material system provides for the establishment and maintenance of mechanized manufacturing standards. As previously described, the minicomputer system maintains control over inventory status and open order status. The master files within the bill of material system and minicomputer system must be compatible in identification and content. This compatability is accomplished by establishing the mini data base as the controlling file and by the transfer of both the bill of material and standard operation routing at the time of a production order release. The transfer occurs either via a request transaction originated on the minicomputer or the fact that a planned order via material requirements planning exists in a specified commited horizon. These conditions cause the bill of material and standard routing to be transmitted to the minicomputer during the periodic execution of the maxi systems.

MATERIAL REQUIREMENTS PLANNING

The material requirements planning system provides the major techniques assisting in the reduction of inventory. The replenishment of inventory is usually controlled via one of two techniques; reorder point or requirements planning. The reorder point technique uses a developed quantity figure based on historical usage which is compared to the on-hand balance when there is activity for the item. If the on-hand balance drops to or below the quantity figure (reorder point), it signals the need to order more of the item. This technique usually causes higher inventory levels at the component item level since higher quantities are stocked in anticipation of the release of a large production order of any of the higher level products which use the component.

The requirements planning technique is based on planned replenishment, when the item is needed. Planned replenishment is accomplished through the establishment of a master schedule for items whose demand is externally controlled. The master schedule is time-phased (spread into planning periods) and usually composed of forecast, customer order backlog, or a combination of both. The system accepts the master schedule and develops a plan based on available and scheduled inventory generating the planned orders (planned replenishment) to accomplish the schedule. In the case where components are needed to manufacture the items, the system calculates the required amount and when they are needed which becomes a portion of their associated master schedule. This calculation is based upon the current bill of material for the item. This planning continues throughout the various bills of material until all items controlled via this technique have been requirements planned. The planning accomplished through the requirements planning system uses various files and parameters. Current inventory and scheduled inventory (on-order) is used to deplete the requirements within the master schedule. The current inventory and scheduled inventory must be accurate to insure a proper materials requirements plan. This information, as previously defined, is maintained on the minicomputer and transmitted to the maxicomputer with any associated master schedule and parameters for a requirements generation execution. The requirements planning technique used is regeneration, thus the total plan is redone once a week.

CAPACITY REQUIREMENTS PLANNING

The capacity requirements planning system supports the identification of production capacity problems. The system accomplishes this by providing visibility of the impact of the production orders (planned and released) on the various work centers within the plant.

The system requires the production order plan generated from the material requirements planning system, a reorder of an item, and any current production order. It then schedules and loads these production orders using the standard operation routings and work center definitions. Output consists of a time-phased report illustrating any overload or underload condition for the various work centers.

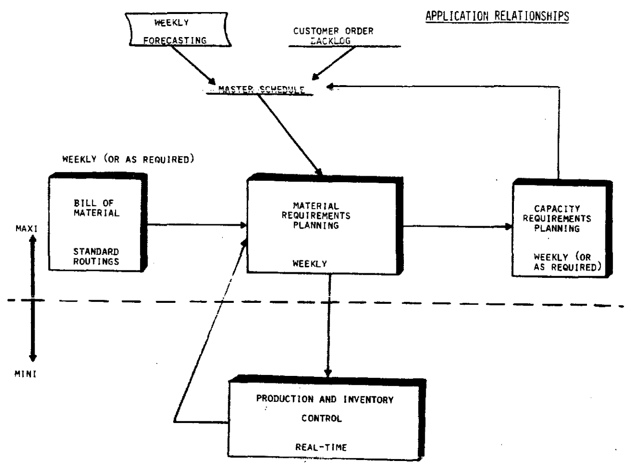


Figure 9

The planned production orders, standard operation routings, and work center definitions exist on the maxicomputer for this periodic run. The current work center load and production order status is maintained on the minicomputer and must be transmitted along with any parameters for subsequent capacity requirements planning.

Figure 9 illustrates the overall relationship of the various application systems and their associated run frequencies. As previously described, the execution of all systems is controlled via the minicomputer system. The customer order backlog and master schedule functions on the chart are currently manual interfaces to the automated systems.

DATA BASE DISTRIBUTION

The distribution of data base responsibility between the two processing units accomplishes a separation of data required to control priorities and capacity versus data required for planning priorities and capacity. The mini data base consists of a set of files which resemble the maxi data base but only contain data required to maintain current inventory, production capacity status and order status.

Figure 10 is an illustration of the mini files and their basic relationships. The inventory master file provides the basic balance information, item identification—description, and control policy information. The open order file contains information about all replenishment orders. Each replenishment order is linked to the appropriate item. The allocation file contains the reservations against the item (via a link). It also provides the linkage which links all the component reservations required to accomplish a production order on the open order file. The allocation records are generated using the bill of material for the item specified on the production order.

The operation detail file provides the various operation steps and extended standards required to produce items specified on the current production orders (linked to open order file). Each operation step to be accomplished in a given work center is linked to the work center file which contains summary load and capacity information. The operation detail records are generated from the standard routing for the item.

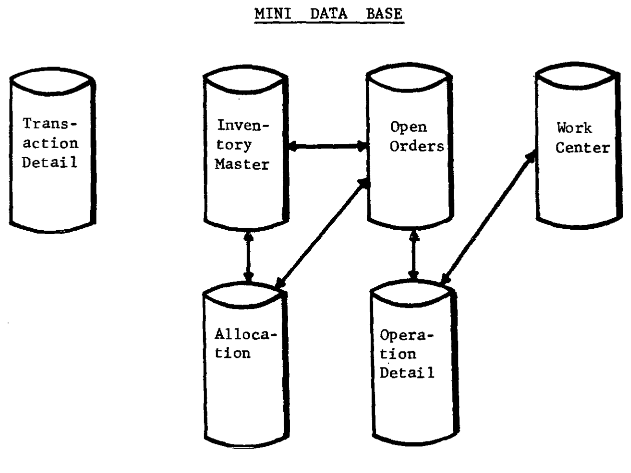


Figure 10

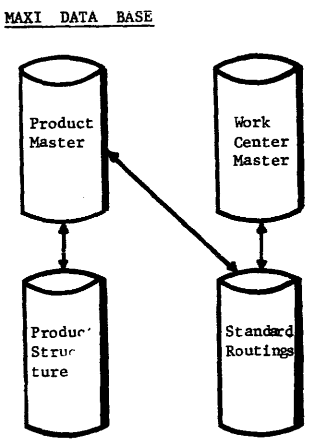


Figure 11

The transaction detail file contains a log of all transactions with associated update information. The file provides a recovery mechanism and also storage of the maxi transactions for subsequent transmission.

Access to the master files is accomplished through the combination of hashing techniques and indices. All files are unordered with linkages providing open space information.

Figure 11 is an illustration of the maxi files and their various relationships. The relationships are similar to those maintained on the mini data base. The product master is a combination of inventory master and open order file as described on the mini. The open order (planned and released) are only stored when the planning systems are executed. The standard routing file equates to the operation detail file on the mini but contains all standard routings for items which can be manufactured. The product structure file equates to the allocation file on the mini but maintains the bill of material for all items which might be produced in the plant. The work center master file is a copy of the work center file on the minicomputer system.

SUMMARY

The installation of this system was accomplished through a time-phased implementation plan with the assistance in design, counseling, and education of G. W. Plossl, Incorporated. The initial implementation consisted of the inventory information modules with the production control modules as subsequent tasks.

The system offers the many intangible benefits which can be attributed to a real-time processing environment. Studies made on the cost alternatives projected that the mini/maxi approach was about equal in cost to the accomplishment of the total system in a batch environment on the maxicomputer.

The maxicomputer alone in a real-time environment increased the costs by one third over the previous alternatives.

The use of this distributed approach to the system provided user involvement throughout the design and implementation cycles. Further development plans may include the use of a minicomputer to handle other major functions, such as, customer order processing. Another potential area is the enhancement of both systems to allow a closer interaction between the mini and maxi computers to provide a net change material requirements planning system.

This paper has presented a brief overview of one system design using a simple distributed system approach. The concept of the system is simple; however, the distribution of the data base and application functions required a detail understanding of the user requirements.

REFERENCES

1. *The Odds are 20 to 1 "PICS" Won't Work for You,* Newsletter Number 8, Oliver Wight, Inc.; P. O. Box 138; Wilton, Connecticut 06897.
2. *What Better Investment Can You Make?,* Newsletter Number 11, G. W. Plossl & Co., Inc.; P. O. Box 32490; Decator, Georgia 30032.
3. *The Production Information and Control System,* IBM; E20-0280-1.
4. *Communications Oriented Production Information and Control System,* IBM; GBOF-4115.

Interactive computer graphics application of the bi-cubic parametric surface to engineering design problems

by G. J. PETERS

McDonnell Douglas Automation Company
St. Louis, Missouri

INTRODUCTION

Engineering design has usually concerned itself with a whole host of mathematical techniques or forms. It is most unusual to expect one device or curve to serve multipurposes. This paper concerns itself with just such a device which may turn out to be a panacea for the engineering designer.

The entity of interest is the parametric cubic (PC) curve and its surface counterpart, the PC patch. The original theoretical analyses have been set forth by Coons[1,2] and Ferguson;[3] Forrest[4] expanded their efforts. Extensive batch computer development of the PC for structural design applications has been performed on the West Coast by Douglas Aircraft and McDonnell Douglas Astronautics—West under the leadership of Eshleman and Meriwether.[5]

The objective at McDonnell Douglas Corporation (MDC) in St. Louis has been to investigate the PC with an eye to complete marriage with the Design/Drafting package which is a completely interactive computer graphics (ICG) system for engineering design purposes as detailed by Lavick[6,7] and Martin.[8]

This paper concerns itself with the mathematical techniques employed in PC work, especially with respect to the PC patch. In addition, actual ICG design examples are shown which reflect the general applicability of the PC to engineering disciplines.

COMPUTER SYSTEM

In the mathematical analyses of the next sections, many figures are presented which depict actual on-line computer communication between man and machine. Therefore, it is deemed necessary to prefix these results with a brief presentation of the computer environment.

Figure 1 depicts the computer hardware and user terminals that are involved in satisfying Computer-Aided Design/Computer-Aided Manufacturing (CAD/CAM) requirements at MDC. Throughout the development of this system, prime consideration has been given to problems of handling large on-line software systems, definition and centralization of a geometric data base, and the ability to provide a satisfactory timesharing environment for economic use of the computer. The specific hardware utilized at MDC-STL is an IBM System 360 (Model 195) computer with four million bytes of main core storage capacity. The reason for such a large facility is that this system is devoted to the application of automation techniques and data processing within a wide range of activities that share resources and cost of this configuration. For example, as can be seen in Figure 1, a large multi-terminal IMS system as well as a general stream of batch processing (scheduled from another large 360 main frame computer) cohabit this configuration with CAD/CAM systems.

This system features a mix of teleprocessing (phone line connected) terminals as well as local connected computer graphics consoles. Thirteen alphanumeric terminals support Production Planning, Tool Design, Quality Assurance, and Loft (including Master Layout) Department activities. They are used for a variety of CAD/CAM functions such as APT part programming, debugging, and data transmission to DNC (Direct Numerical Control) IBM/1800 satellite computers for on-line machining or on-line drafting. Nine of the IBM 2250 vector graphics terminals are located in user project areas at distances up to 12,000 feet from the computer, which is believed to be the farthest remote interactive facility in the world. Figure 2 illustrates the layout of these terminals and their relative distance with respect to the location of the main frame computer. The philosophy and effectiveness of locating these types of devices in direct user areas are highlighted in References 7 and 8. One CRT console, located in the Loft engineering department, is used primarily for Loft surface shape definition. Two consoles are located in the Manufacturing Production Planning department. These are used in the Graphics Numerical Control (GNC) process which involves retrieving an engineering drawing and then interactively directing the machine tool around the drawing to generate the APT part program. Six scopes, located in Engineering project areas, are dedicated to structural/mechanical design applications.

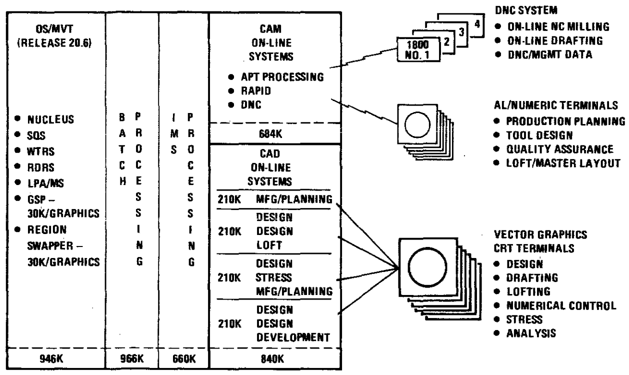


Figure 1—Interactive CAD/CAM configuration at St. Louis (May 1973)

PARAMETRIC REPRESENTATION

Reference 4 presents many reasons for using the parametric form of a curve vis-a-vis the non-parametric form. A few of them will now be discussed.

The use of the non-parametric form is largely confined to planar curves. Aircraft and ship lofting use conic sections, cubics, and higher order polynomials, and lofting techniques for these curves are well.established. To avoid some of the difficulties inherent in the non-parametric form, Newell[9] has classified the conic into 34 separate cases and Shelly[10] using a different approach into three cases. By their very nature, conics cannot yield curves with points of inflection. Yet such curves very often exist in aircraft shapes, for example, in duct contours.

The planar cubic in the non-parametric form arises mainly from the use of splines. The non-parametric spline is a two-dimensional curve. It is continuous in both first and second derivatives throughout its length, and is considered by the aircraft and shipbuilding industries to give a smooth and fair curve. There is an inherent difficulty when curve fitting with splines where the data has an implied discontinuity in second derivative. While this is not too common, in theory at least, in the case of ships and aircraft, it is a frequent occurrence in mechanical design. An obvious example is the fillet radius joining two straight lines: in this case the spline tends to oscillate about the expected curve in an unacceptable manner[11] unless appropriate end conditions are applied. In addition, splines may be inconvenient when the deflections become large or the radius of curvature becomes small.

In general, the non-parametric curve form has several inherent drawbacks. A curve which is defined by tangent properties as well as points may well require that the slope be infinite. This can be avoided either by changing the coordinate axes or by using a different form of equation, both of which are cumbersome procedures. Curve segments must be bounded by defining the end points but the tests to determine whether a point lies on the bounded segment can be elaborate and even ambiguous when the curve loops. In the case of two-dimensional curves, a given value of x may yield several values of y which must be tested, and in the case of twisted curves the difficulty is compounded. This complicates the computation for display, plotting, etc., of points on the curve; computation may involve evaluating square, cube and higher roots. If the curve is to be plotted either as a series of points or a series of straight lines, the computation involved to generate a visually smooth curve could be very great. Parametric methods overcome many of these difficulties.

Consider the parametric representation of a general continuous curve in 3-space as a transformation of the form

$$x=f(u), \qquad y=g(u), \qquad z=h(u)$$

defined for u in the interval $[a, b]$. In vector notation,

$$\overline{r(u)} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f(u) \\ g(u) \\ h(u) \end{pmatrix}$$

then

$$\frac{d\bar{r}}{du} = \begin{pmatrix} dx/du \\ dy/du \\ dz/du \end{pmatrix} = \begin{pmatrix} f'(u) \\ g'(u) \\ h'(u) \end{pmatrix}$$

is the tangent vector. The real slopes of the curve are given by the ratios of the components of the tangent vector. For example,

$$\frac{dz}{dx} = (dz/du)/(dx/du) = \frac{h'(u)}{f'(u)} .$$

An infinite slope is specified by setting one component of the tangent vector to zero.

A parametric curve is bounded by two parametric values. The test for a point lying on the curve reduces to finding the parametric value defining the point and checking that this value lies in the stated range. Computation of points on the curve segment is by substitution of a parametric value in two

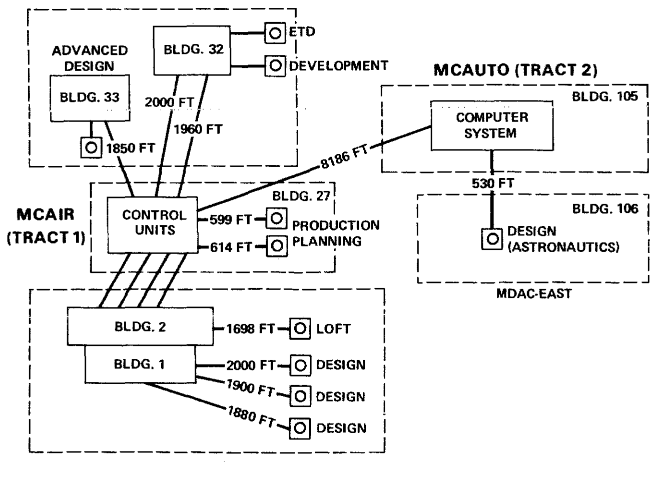


Figure 2—Remote locations of on-line computer graphics terminals at St. Louis (May 1973)

or three equations which in the case of commonly used curves such as conics and cubics will be polynomials rather than equations involving roots. If the curve is twisted, it is clearly easier to substitute one value of a parameter in three equations and obtain x, y, and z directly than to solve for points lying on the intersections of two surfaces and check that they lie on the specified portion of the curve. The parametric form of the curve is easily transformed into a curve of similar form but different orientation or scale by matrix multiplication; that is to say the mathematical form of the curve may be axis independent. Similar transforms on non-parametric curves are more difficult.

Hence, the parametric form is not only more general, but it is very well suited to computation and display. In addition, it will be shown that this form has properties which are attractive for computer-aided design.

PARAMETRIC CUBIC CURVE

Mathematical form

The PC space curve is represented as:

$$V(u)=Au^3+Bu^2+Cu+D \tag{1}$$

where V is defined as a general coordinate and stands for x, y, or z.

In matrix notation

$$V(u)=(u^3\ u^2\ u\ 1)(A\ B\ C\ D)^T. \tag{2}$$

where T is the transpose of the row matrix.

For convenience, the curve parameter u is defined to be in the region $0\leq u\leq 1$.

Equation (1) is called the algebraic form of the PC since it is the usual polynomial representation of a function.

The parametric derivative is

$$\frac{dV}{du}=V'(u)=3Au^2+2Bu+C \tag{3}$$

or

$$\frac{dV}{du}=(3u^2\ 2u\ 1\ 0)(A\ B\ C\ D)^T. \tag{4}$$

Consider the following set of 4 equations obtained by setting $u=0$ and $u=1$ in (1) and (3)

$$V(0)=D \tag{5}$$

$$V(1)=A+B+C+D \tag{6}$$

$$V'(0)=C \tag{7}$$

$$V'(1)=3A+2B+C \tag{8}$$

or

$$\begin{pmatrix} V(0) \\ V(1) \\ V'(0) \\ V'(1) \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix}. \tag{9}$$

Solving the linear system (9) for the algebraic coefficients leads to

$$(A\ B\ C\ D)^T=(M)(V(0)V(1)V'(0)V'(1))^T \tag{10}$$

where

$$M=\begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{11}$$

Substituting (10) into (1), and re-arranging, yields

$$V(u)=V(0)F_1(u)+V(1)F_2(u)+V'(0)F_3(u)+V'(1)F_4(u) \tag{12}$$

where

$$F_1(u)=2u^3-3u^2+1$$

$$F_2(u)=-2u^3+3u^2$$

$$F_3(u)=u^3-2u^2+u$$

$$F_4(u)=u^3-u^2. \tag{13}$$

Equation (12) is called the geometric form of the PC curve and (13) defines the so-called "blending functions" which serve the purpose of blending the quantities $V(0)$, $V(1)$, $V'(0)$, and $V'(1)$ together so as to form a continuous curve satisfying the end conditions.

In matrix notation, we can write (12) as

$$V(u)=(u^3\ u^2\ u\ 1)(M)\begin{pmatrix} V(0) \\ V(1) \\ V'(0) \\ V'(1) \end{pmatrix}. \tag{14}$$

Note that the end position values and parametric derivative values of the curve are obtained by inspection of (14). For this reason, the geometric form is sometimes preferred over the algebraic form (2).

Expanding (14), the curve is totally represented as

$$\begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix}^T=(u^3\ u^2\ u\ 1)(M)\begin{pmatrix} x(0) & y(0) & z(0) \\ x(1) & y(1) & z(1) \\ x'(0) & y'(0) & z'(0) \\ x'(1) & y'(1) & z'(1) \end{pmatrix}. \tag{15}$$

Figure 3 depicts the correspondence between real x, y, z space and parametric space and, in addition, shows the parametric plot. It is most important to realize that usually x, y, and z can only be related through the parameter u.

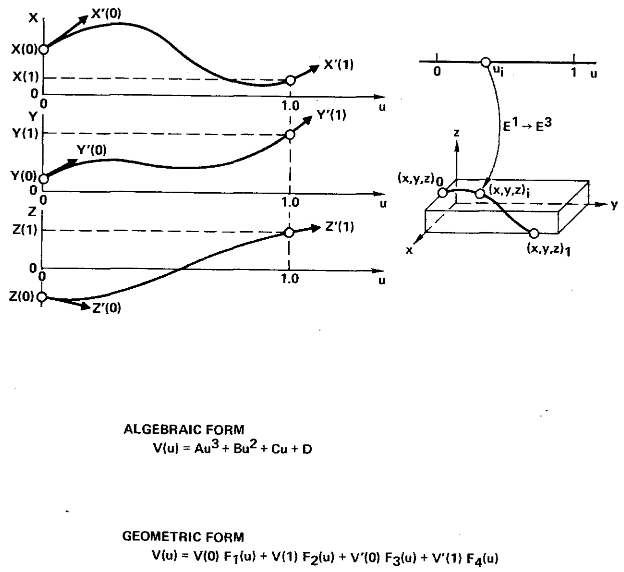


Figure 3—Parametric cubic (PC) space curve-mathematical form

Creating a PC curve

There are many ways to create a PC curve; however, each way must provide sufficient data to determine the 12 coefficients implied by (1) or (14) i.e., 4 coefficients for x, y, and z respectively. From a numerical analysis viewpoint, the following are three important considerations:

(1) If only position data points are given, are the interior u values specified?
(2) If direction cosine data are given in addition to point data, be careful concerning the dependency of the real slope data.
(3) Are the given data planar or non-planar (twisted)? A few cases will now be considered.

Case 1. Witness the data: 4 points (planar or twisted) with the interior u values specified, i.e., $0<u_1<u_2<1$. Using (1) or (12), the unique PC can be obtained by solving 4 linear simultaneous equations for each coordinate x, y, z.

Case 2. Witness the data: 3 points with non-planar direction cosines at the first and last point. Using the geometric form

$$V(u)=V(0)F_1(u)+V(1)F_2(u)+V'(0)F_3(u)+V'(1)F_4(u)$$

The unknowns are the internal value of u corresponding to the middle point and the parametric tangents, $V'(0)$ and $V'(1)$. It is now necessary to relate the direction cosines to the parametric tangents. Consider the tangent vector

$$\bar{T}=\left(\frac{dx}{du}\ \frac{dy}{du}\ \frac{dz}{du}\right)^T \tag{16}$$

then

$$S(u)=\|\bar{T}\| \tag{17}$$

is the length of the tangent vector at point u. Furthermore

$$\frac{dx}{du}=S(u)a$$

$$\frac{dy}{du}=S(u)b$$

$$\frac{dz}{du}=S(u)c \tag{18}$$

and (18) relates the parametric tangents to the direction cosines a, b, and c at the point of interest. Re-writing our system to be solved for the unique PC

$$x(u)=x(0)F_1(u)+x(1)F_2(u)+S(0)a_0F_3(u)+S(1)a_1F_4(u)$$

$$y(u)=y(0)F_1(u)+y(1)F_2(u)+S(0)b_0F_3(u)+S(1)b_1F_4(u)$$

$$z(u)=z(0)F_1(u)+z(1)F_2(u)+S(0)c_0F_3(u)+S(1)c_1F_4(u) \tag{19}$$

These three non-linear equations in the three unknowns can be solved very efficiently by N-dimensional Newton-Raphson [12] provided a judicious choice is made for the initial solution vector $((uS(0)S(1))^T)_0$. Then (18) is used to obtain the parametric tangents which together with the given position data determine a unique PC. Note that it was necessary to obtain the lengths of the tangent vectors to establish uniqueness since many curves can exhibit the same direction cosines at their end points.

Figure 4 shows some examples of PC creation at the CRT. These curves do not represent an over-conditioned situation; they were created by conjuring up the minimum data required for a unique PC.

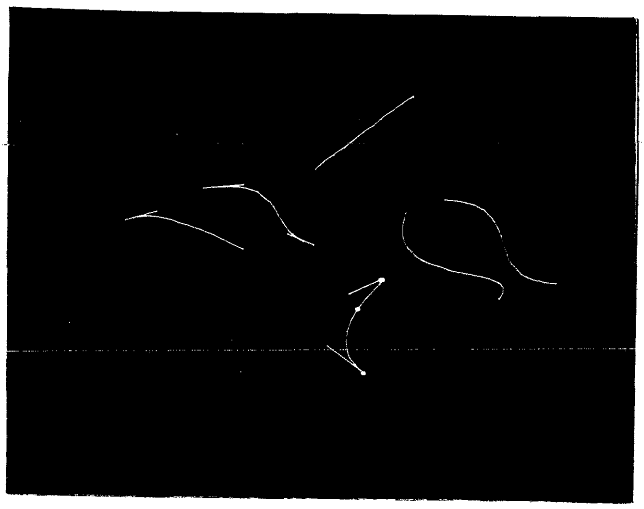


Figure 4—Creating planar or non-planar PC curves at the CRT

Planar/Twist PC Test

The "measure of twist" or torsion of a space curve is obtained by using Frenet's formulas which are fundamental in the theory of space curves. It can readily be shown [13] that the torsion is given by

$$\tau = \frac{\left(\dot{\bar{r}} \times \ddot{\bar{r}}\right) \cdot \dddot{\bar{r}}}{\left|\dot{\bar{r}} \times \ddot{\bar{r}}\right|^2} \tag{19}$$

where

$$\bar{r} = (x(u)y(u)z(u))^T \tag{20}$$

is the position vector to any point on the curve and the dot above the vector signifies differentiation with respect to u. Assuming the denominator in (19) is not zero (the straight line PC), the condition for a plane curve or zero torsion is

$$\left(\dot{\bar{r}} \times \ddot{\bar{r}}\right) \cdot \dddot{\bar{r}} = 0 \tag{21}$$

Moreover, (21) is the scalar triple product (or "box product") which can be expressed in determinant form as

$$\begin{vmatrix} \dot{x} & \dot{y} & \dot{z} \\ \ddot{x} & \ddot{y} & \ddot{z} \\ \dddot{x} & \dddot{y} & \dddot{z} \end{vmatrix} = 0. \tag{22}$$

Now, using the algebraic form of the PC, taking the required parametric derivatives, and substituting in (22) leads to

$$\begin{vmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{vmatrix} = 0 \tag{23}$$

Thus, to ensure no twist, the coefficients of the PC must satisfy (23). Observe that if A_x, A_y, and A_z are all zero, then the torsion is zero. This means, of course, that the "space curve" is a parametric quadratic; therefore, parametric quadratics can have no twist and thus are planar curves. Hence, the minimum power required for a parametric representation to be twisted is three, i.e., cubic. Also, if all the quadratic coefficients (B_x, B_y, B_z) or all the linear coefficients (C_x, C_y, C_z) are zero, then the curve is also planar.

Effect of varying the tangent vector length

A powerful feature of parametric representation is the ability to drastically change the shape of the curve by varying the tangent vector length, S, at the end points while maintaining the end direction cosine slopes. For example

$$\frac{dy}{dx} = \frac{dy/du}{dx/du} = \frac{K\,dy/du}{K\,dx/du} \tag{24}$$

where K is an arbitrary constant. Note that it is very easy to handle an infinite slope for $\frac{dy}{dx}$; set $\frac{dx}{du}$ equal to zero.

Interactively, S variation can be used as a design tool to modify an existing PC, e.g., to clear an obstruction. Figure 5 shows the effect of S variation and indicates that unwanted kinks can occur if the K factor becomes too large.

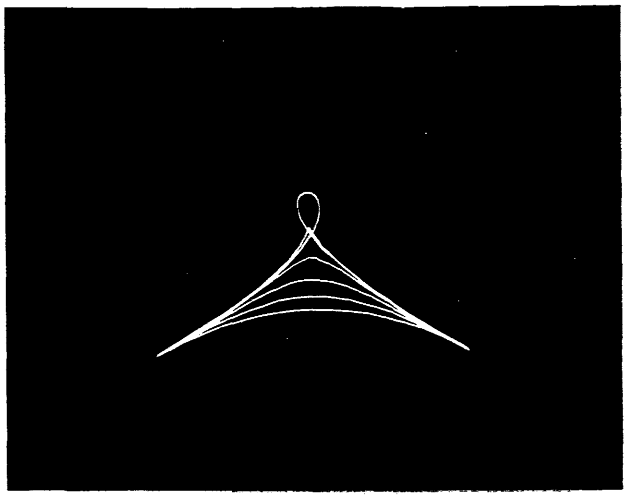


Figure 5—Effect of varying PC tangent vector length

Arc length

In parametric form, PC arc length is expressed as

$$L = \int_{u_1}^{u_2} \sqrt{\dot{\bar{r}} \cdot \dot{\bar{r}}}\, du \tag{25}$$

where $u_2 > u_1$ and $\bar{r}$ is from (20). After simplifying, (25) can be written as

$$L = \int_{u_1}^{u_2} \sqrt{A_4u^4 + A_3u^3 + A_2u^2 + A_1u + A_0}\, du \tag{26}$$

where the A_i constants are defined in terms of the algebraic PC coefficients. Functionally, (26) may be expressed as

$$L = \int_{u_1}^{u_2} f(u)\, du \tag{27}$$

Using Gauss quadrature [14],

$$\int_{u_1}^{u_2} f(u)\, du = \sum_{i=1}^{N} w_i f(u_i) \tag{28}$$

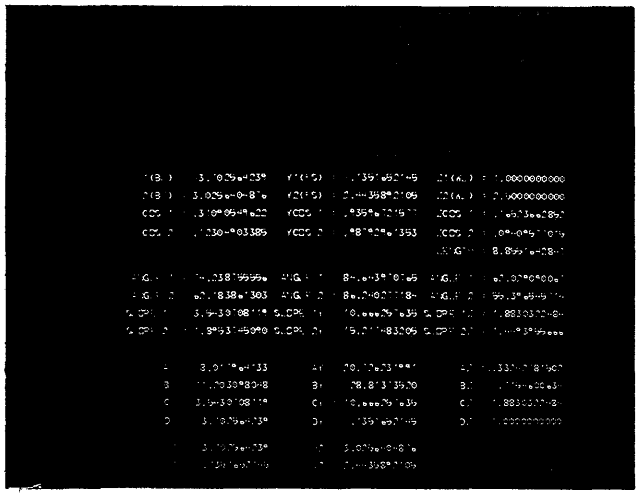


Figure 6—Geometric properties of the PC

where N is the number of points used, w_i are the weight values, and u_i are the Gaussian abcissae. The Gaussian abcissae may be normalized to a more convenient interval $(0 \leq u_i \leq 1)$ by using the transformation

$$z = \frac{u - u_1}{u_2 - u_1} \tag{29}$$

Then, L may be written

$$L = (u_2 - u_1) \int_0^1 f(u_1 + (u_2 - u_1)z)\, dz \tag{30}$$

or

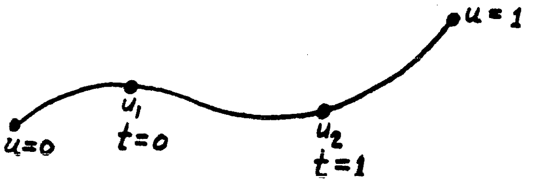
$$L = (u_2 - u_1) \sum_{i=1}^{N} w_i g(z_i) \tag{31}$$

where weights and abcissae are with respect to the new interval. For example, using $N = 9$ with the weights and abcissae obtained from [15], the quadrature formula possesses a degree of precision of $2N - 1$, i.e., yields exact results if $f(u)$ would be a 17th order polynomial or less.

The basic geometric properties of the PC, including arc length $(N = 9)$, are shown in Figure 6.

Segment a PC

Consider a given curve. It is desired to define a new curve from u_1 to u_2 on the original curve. See the sketch.



Using the linear transformation

$$u = u_1 + t(u_2 - u_1) \tag{32}$$

$$\frac{du}{dt} = u_2 - u_1. \tag{33}$$

At $t = 0$, $u = u_1$; at $t = 1$, $u = u_2$. Since $V(u) = V(u(t))$

$$\frac{dV}{dt} = \frac{dV}{du}\frac{du}{dt} \tag{34}$$

Using (34) and denoting the new curve by $v(t)$, the geometric coefficients of the segmented curve in terms of the given curve are

$$\begin{bmatrix} v(0) \\ v(1) \\ v'(0) \\ v'(1) \end{bmatrix} = \begin{bmatrix} V(u_1) \\ V(u_2) \\ (u_2 - u_1) V'(u_1) \\ (u_2 - u_1) V'(u_2) \end{bmatrix} \tag{35}$$

Several examples of segmenting a given curve are shown in Figure 7.

Translation, rotation, and scaling

The transformation operations on the PC are actually performed on equations.

For translation

$$v(u) = V(u) + \Delta V \tag{36}$$

where $v(u)$ is the resultant x, y, or z coordinate and ΔV is the respective translation. In algebraic form, the translation is added to the D coefficient; in geometric form, it is added to both $V(0)$ and $V(1)$.

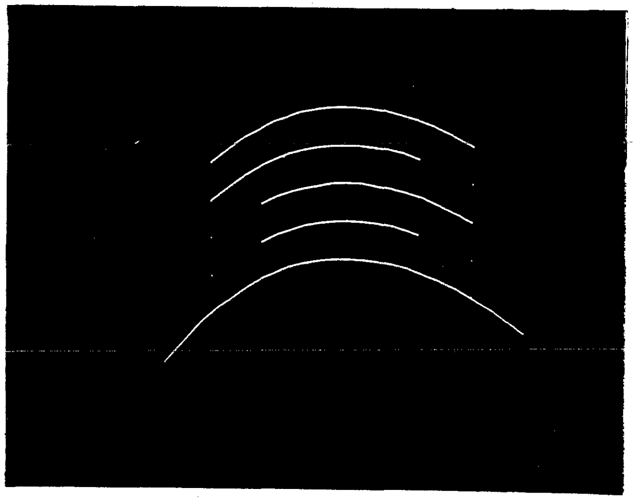


Figure 7—Segmenting a PC curve

For rotation

$$\begin{pmatrix} X(u) \\ Y(u) \\ Z(u) \end{pmatrix} = (A) \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix} \tag{37}$$

where (A) is the 3×3 rotation matrix. If the given PC is in geometric form,

$$\begin{pmatrix} X(u) \\ Y(u) \\ Z(u) \end{pmatrix}^T = (u^3\ u^2\ u\ 1)(M) \begin{bmatrix} x(0) & y(0) & z(0) \\ x(1) & y(1) & z(1) \\ x'(0) & y'(0) & z'(0) \\ x'(1) & y'(1) & z'(1) \end{bmatrix} (A) \tag{38}$$

For scaling

$$\begin{pmatrix} X(u) \\ Y(u) \\ Z(u) \end{pmatrix} = \begin{pmatrix} 1-k_1 & 0 & 0 \\ 0 & 1-k_2 & \\ 0 & 0 & 1-k_3 \end{pmatrix} \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} + \begin{pmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{pmatrix} \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix} \tag{39}$$

where k_1, k_2, and k_3 are the scale factors for x, y, and z, respectively, and the subscript r signifies a reference point for scaling. For equal scaling, sometimes called ratio, (39) can be written

$$\begin{pmatrix} X(u) \\ Y(u) \\ Z(u) \end{pmatrix} = (1-k) \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix} + k \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix} \tag{40}$$

where k is the common scale factor.

Figure 8 shows PC curve transformations.

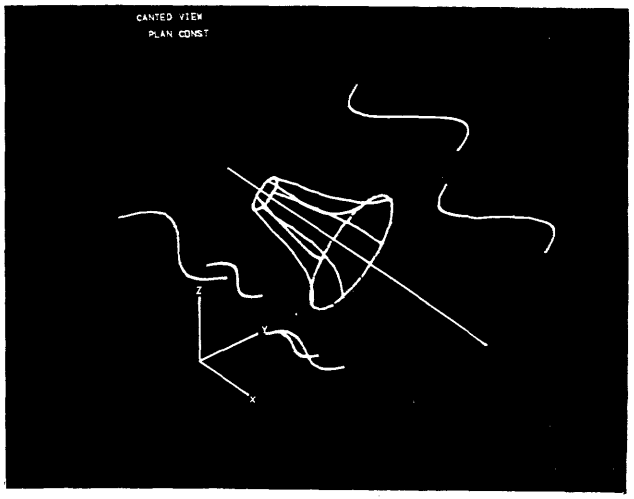


Figure 8—Translation, rotation and scaling of Pc curves

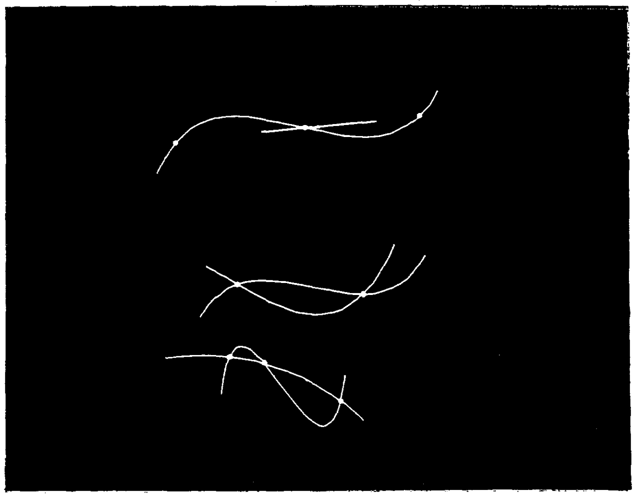


Figure 9—Intersecting a PC with a plane or a PC

Intersections

Two cases will be considered: the intersection of a plane with a PC, and the intersection of two PC curves.

For the former case, let the plane be determined by the normal vector $\bar{N}$ and a point $\bar{p}_0$ in the plane. At the intersection, the point $\bar{p}$ will lie on the PC and also in the plane; therefore, using the definition of a plane

$$\bar{N} \cdot \overline{(\bar{p}-\bar{p}_0)} = 0 \tag{41}$$

or

$$N_x(x(u)-x_0)+N_y(y(u)-y_0)+N_z(z(u)-z(0))=0. \tag{42}$$

Since each coordinate is a cubic in u, (42) is a cubic equation which can be solved (either closed form or iteratively) for its real roots. Of course, the roots of interest are $0 \leq u_i \leq 1$.

For the latter case, the following realistic assumptions are made: (a) each PC is planar and (b) both curves are coplanar. At a point of intersection

$$\overline{r_{pc1}} = \overline{R_{pc2}} \tag{43}$$

where $\bar{r}$ and $\overline{\text{R}}$ are the position vectors of the intersection point with respect to curves 1 and 2, respectively. Therefore, since the problem is planar

$$\begin{aligned} x(u)-X(w)=0 \qquad 0 \leq u \leq 1 \\ y(u)-Y(w)=0 \qquad 0 \leq w \leq 1 \end{aligned} \tag{44}$$

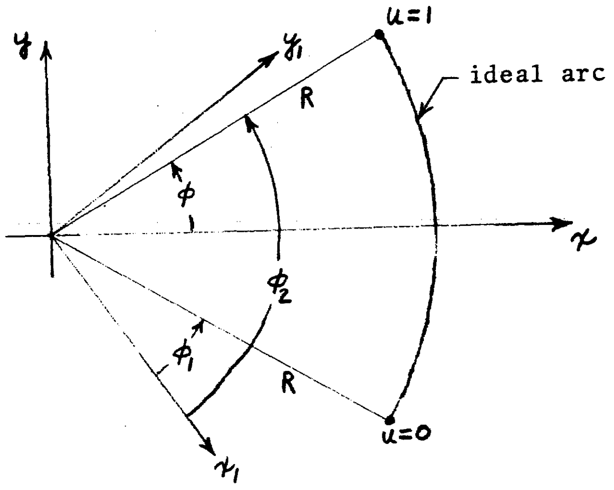
Expanding the x component equation and using the algebraic form

$$A_x u^3 + B_x u^2 + C_x u + D_x - (A_X w^3 + B_X w^2 + C_X w + D_X) = 0 \quad (45)$$

A similar equation can be written for y. Hence the result is two non-linear equations in two unknowns, u and w. Since a light pen detect can be made very close to the visible intersection on the CRT, Newton-Raphson is a very efficient servant for this problem.* Some CRT examples are shown in Figure 9.

Converting an arc to a PC

Consider the following picture



Using the above geometry, and assuming a unit radius the approximating PC can be expressed as

$$x(u) = (u^3\ u^2\ u\ 1)(M) \begin{bmatrix} \cos\phi \\ \cos\phi \\ 4(1-\cos\phi) \\ -4(1-\cos\phi) \end{bmatrix} \quad (46)$$

$$y(u) = (u^3\ u^2\ u\ 1)(M) \begin{bmatrix} -\sin\phi \\ \sin\phi \\ 4(1-\cos\phi)/\tan\phi \\ 4(1-\cos\phi)/\tan\phi \end{bmatrix} \quad (47)$$

where

$$\phi = (\phi_2 - \phi_1)/2.$$

The question of accuracy necessarily comes to mind. How good is the approximation? At any point $0 \le u \le 1$,

$$r(u) = \sqrt{(x(u))^2 + (y(u))^2} \quad (48)$$

where $r(u)$ is the distance from the origin to the PC. The error function is

$$\Delta R(u) = r(u) - R \quad (49)$$

where R is the radius of the true arc ($R=1$ in (46) and (47)). Normalizing with respect to the given radius

$$\frac{\Delta R(u)}{R} = \frac{r(u)}{R} - 1. \quad (50)$$

For a given ϕ, the task is to extremize $\Delta R(u)/R$ since this would provide the worst error to characterize the fit; therefore

$$\frac{\Delta R}{R} = \underset{u \in [0,1]}{\text{ext}} \left(\frac{\Delta R(u)}{R} \right) \quad (51)$$

and the values of u are the extrema of

$$\frac{d}{du}(r(u)) = 0. \quad (52)$$

Performing the differentiation yields

$$x(u)\frac{dx}{du} + y(u)\frac{dy}{du} = 0 \quad (53)$$

which is a 5th order equation in u. By inspection of (46) and (47), there are three u values for which ΔR is zero; these values are 0, 0.5, and 1. In addition, these u's also satisfy (53). Therefore, solving the resultant quadratic equation yields

$$u = \tfrac{1}{2} \pm \tfrac{1}{2}\sqrt{1 - 4(k/k_2)} \quad (54)$$

where k and k_2 are functions of the given ϕ only. Note that the extrema deviations are symmetric with respect to $u=0.5$. Using the values from (54) in (46) and (47), it can be shown that the error excursion about the true arc is always positive. Furthermore, these two values produce equal deviations.

Typical arc-to-PC examples are shown in Figure 10 for radii of 20, 30, 40, and 50 inches. The left portion shows the

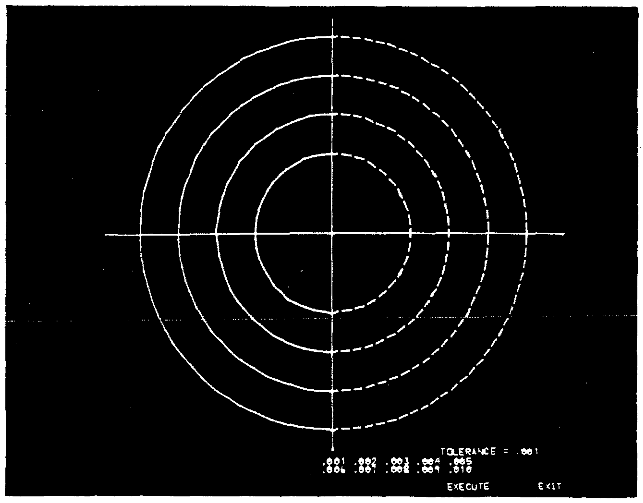


Figure 10—Converting an arc or circle to a PC

* According to Acton,[12] Newton-Raphson is rather like the little girl in the nursery rhyme who had a curl in the middle of her forehead and "when she was good, she was very, very good; but when she was bad, she was horrid!"

original circles while the right portion shows the PC approximations. The number of PC curves are determined by keying in the desirable tolerance ΔR. Using $\Delta R = .001$ inch with the above radii produced 3 PC curves for the 180 degree subtended angle shown in the figure. However, with $R=5$ inches and the same tolerance, 2 PC curves would be created. Figure 11 shows the maximum error obtained for various total arc angles.

Curves do not occur in isolation in the description of solid objects, but bound or lie on the surface regions which define these objects. Thus, the study of curves is a prerequisite to the study of surfaces. Therefore, having covered some of the features of the PC curve, it is now possible to discuss its surface extension, the PC patch.

PARAMETRIC CUBIC SURFACE PATCH

Mathematical form

The general continuous surface in 3-space can be expressed as a transformation of the form

$$x = f(u, w)$$
$$y = g(u, w)$$
$$z = h(u, w)$$

whose domain is a set D in the uw plane. In particular, the PC surface patch is defined as

$$V(u, w) = (u^3\ u^2\ u\ 1)(M)(B)(M)^T \begin{pmatrix} w^3 \\ w^2 \\ w \\ 1 \end{pmatrix} \tag{55}$$

where (M) was defined in (11) and

$$(B) = \begin{pmatrix} V_{00} & V_{01} & V_{00w} & V_{01w} \\ V_{10} & V_{11} & V_{10w} & V_{11w} \\ V_{00u} & V_{01u} & V_{00uw} & V_{01uw} \\ V_{10u} & V_{11u} & V_{10uw} & V_{11uw} \end{pmatrix} \tag{56}$$

θ (DEGREES)	$\frac{\Delta R}{R}$
10	4.2×10^{-10}
20	2.7×10^{-8}
30	3.1×10^{-7}
45	4.0×10^{-6}
60	2.0×10^{-5}
90	2.5×10^{-4}
120	1.3×10^{-3}
160	7.4×10^{-3}
180	1.5×10^{-2}

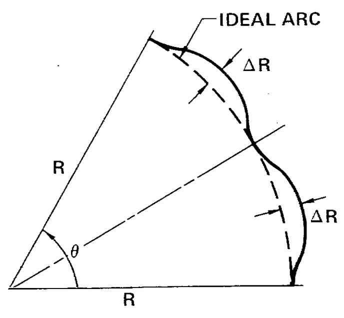
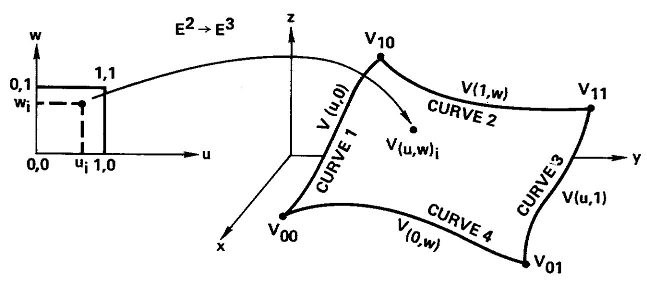


Figure 11—ARC to PC conversion accuracy



GEOMETRIC FORM

$$V(u,w) = [u^3\ u^2\ u\ 1]\ [M] \underbrace{\begin{bmatrix} V_{00} & V_{01} & V_{00w} & V_{01w} \\ V_{10} & V_{11} & V_{10w} & V_{11w} \\ V_{00u} & V_{01u} & V_{00uw} & V_{01uw} \\ V_{10u} & V_{11u} & V_{10uw} & V_{11uw} \end{bmatrix}}_{[B]} [M]^T \begin{bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{bmatrix}$$

ALGEBRAIC FORM

$[S] = [M]\ [B]\ [M]^T$ = ALGEBRAIC FORM OF PATCH COEFFICIENTS

Figure 12—PC patch mathematical form

The matrix B is called the boundary matrix since its elements are geometric properties of the boundaries of the surface patch. Figure 12 summarizes the basic concepts of a patch.

It is important to understand the notation in (56). The number subscripts refer to values of the parametric variables u and w at the corner points while the letter subscripts indicate the derivative with respect to that parametric variable. For example

$$V_{01} = [V(u, w)]_{u=0, w=1} = V(0, 1) \qquad \text{point data} \tag{57}$$

$$V_{10w} = \left[\frac{\partial V(u, w)}{\partial w}\right]_{u=1, w=0} = V(0, 1)_w \qquad \text{rate data} \tag{58}$$

$$V_{11uw} = \left[\frac{\partial^2 V(u, w)}{\partial u\, \partial w}\right]_{u=1, w=1} = V(1, 1)_{uw} \qquad \text{twist data} \tag{59}$$

Recall that V stands for x, y, or z. There is a B matrix for each coordinate. Using a more compact matrix notation

$$V(u, w) = (F(u))(B)(F(w))^T \tag{60}$$

where

$$(F(u)) = (u^3\ u^2\ u\ 1)(M) \tag{61}$$

and

$$(F(w)) = (w^3\ w^2\ w\ 1)(M) \tag{62}$$

are the blending functions defined in (13).

Computing expense is minimized by the use of another form

$$V(u, w) = (U)(S)(W)^T \tag{63}$$

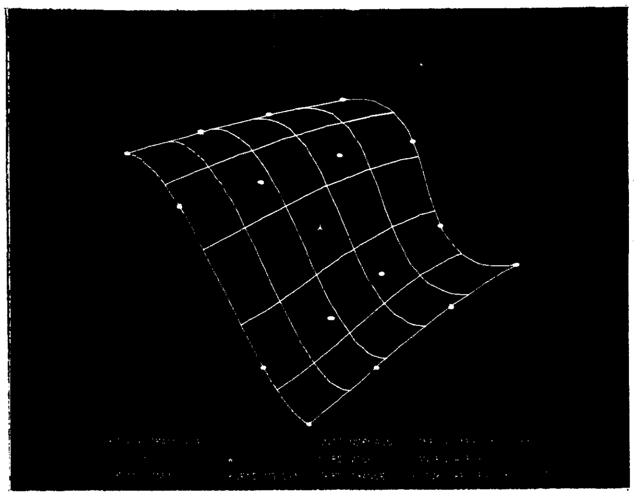


Figure 13—PC patch defined by 16 points

where

$$(U)=(u^3\ u^2\ u\ 1) \tag{64}$$

$$(W)=(w^3\ w^2\ w\ 1) \tag{65}$$

and

$$(S)=(M)(B)(M)^T. \tag{66}$$

Analogous to the PC curve, the terms "geometric" and "algebraic" form also apply to the PC patch coefficients. Boundary matrix (B) of (56) is the geometric form while the surface matrix (S) is called the algebraic form.

Equation (56) now requires more attention since it embodies the individual geometric character of any surface patch. Note that the first and second rows are the PC boundary curves for $u=0$ and $u=1$, respectively; while the first and second columns are the curves for $w=0$ and $w=1$, respectively. Following Herzog,[16] (56) can be divided into four 2×2 partitions

$$(B)=\begin{pmatrix} P & R_w \\ R_u & T \end{pmatrix}. \tag{67}$$

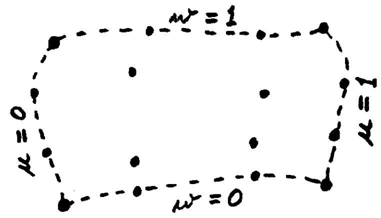
The partition P contains the position data of the four end or corner points. Partitions R_u and R_w contain parametric rates which are related to the real slopes or tangents at the corner points. The interior character of the patch is controlled by the twist or cross-derivative partition, T. It should be emphasized that the richness of information contained in matrix (B) is readily combined and manipulated by the computer.

From a terminology point of view, the terms "surface" and "patch" are used interchangeably. However, in a more general sense, surface is the superset since a surface can contain one or more patches.

Creating a PC patch

Similarly as the PC curve, there are many ways to create a PC patch; however, each way must provide sufficient data to determine the 48 coefficients implied by (56) or (66), i.e., 16 coefficients for x, y, and z, respectively. Some specific cases will now be considered.

Case 1. Witness the data: A grid of 16 points (planar or twisted) with uw values not specified in advance. See the sketch with a rough outline of the desired boundary curves through the border points.



An estimate of the uw values for other than the corner points can be obtained by using a ratio of line segments to get a value between 0 and 1. Then using the algebraic form of the patch in (63)

$$(u^3\ u^2\ u\ 1)(S)\begin{pmatrix} w^3 \\ w^2 \\ w \\ 1 \end{pmatrix}=V(u,w) \tag{68}$$

and expanding this

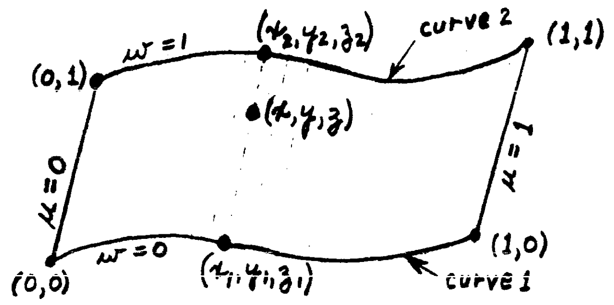
$$(u^3w^3)s_{11}+(u^3w^2)s_{12}+(u^3w)s_{13}+(u^3)s_{14}+(u^2w^3)s_{21}+\cdots$$
$$+(uw^3)s_{31}+\cdots+(w^3)s_{41}+\cdots+s_{44}=V(u,w) \tag{69}$$

or

$$(C)(\bar{p})=(\bar{b}). \tag{70}$$

Hence, the result is a linear simultaneous system of equations whose unknowns are the elements of (S). Specifically, (C) is a 16×16 matrix of uw pairs; $(\bar{p})$ is a 16×1 vector of the unknown (S) elements; and $(\bar{b})$ is a 16×1 vector of the given data points. Note that (70) must be solved for x, y, and z. The Gauss-Jordan elimination with maximum pivot strategy[17] readily handles this problem. Figure 13 depicts a resultant patch created at the CRT.

Case 2. Witness the data: Two PC curves (planar or twisted). Create a ruled surface patch between them. Consider the following sketch

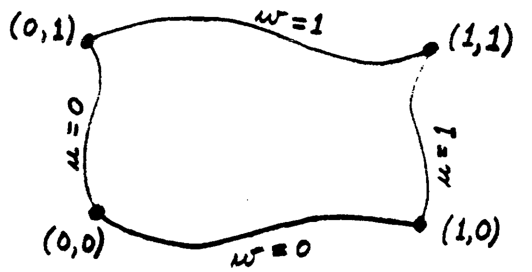


and assume that the surface is to be ruled in w; therefore, curves 1 and 2 are $f(u)$. It can be shown that the general point on the patch is given by

$$V(u,w)=V_1(u)+w(V_2(u)-V_1(u)) \qquad 0\leq w\leq 1 \quad (71)$$

where the subscripts refer to curves 1 or 2. Since the parametric derivative and cross derivative can easily be obtained from (71), all the elements of the (B) matrix are known.

Case 3. Witness the data: Four connected PC curves (planar or twisted). See the sketch.



This case is the classic Coons patch with zero twist elements in the (B) matrix. The patch is completely defined by the four given boundary curves.

Figure 14 shows planar and non-planar ruled patches and Coons patches.

Determination of cross-derivatives

The cross-derivative elements of the boundary matrix are essential to the PC patch system. They not only provide higher accuracy in fitting given data but also permit adjustment of the surface normal slope along a boundary. Therefore, they have very practical significance.

The questions come to mind: How are the cross-derivatives originally determined? Is there a general equation for them? There is no general rule. Each surface has its own characteristic interior shape; many times this is dictated by the designer. For example, if a ruled surface is desired, using (71) yields

$$\frac{\partial^2 V(u,w)}{\partial u\,\partial w}=\frac{dV_2(u)}{du}-\frac{dV_1(u)}{du} \qquad (72)$$

and evaluating (72) for the corner values of u and w determines the four numbers to be placed in partition T of matrix (B).

Reference 16 discusses methods for adjusting T while Reference 5 presents some excellent pictures showing the effect of the cross derivatives on the patch interior.

At MDC-St. Louis, the interior of the patch can also be modified interactively in two ways: (1) using a tracking cross to move selected defining points to other desirable locations and (2) keying in new values for elements of T.

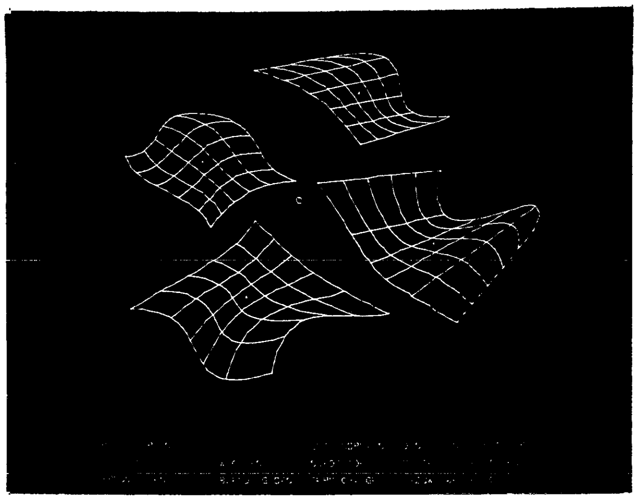


Figure 14—PC ruled patches and coons patches

Normal vector

For a smooth surface in 3-space, the normal vector at a point is

$$\bar{N}=\bar{r}_u\times\bar{r}_w=\left(\frac{\partial\bar{r}(u,w)}{\partial u}\right)\times\left(\frac{\partial\bar{r}(u,w)}{\partial w}\right) \qquad (73)$$

or using the Jacobian notation

$$\bar{N}=\left(\frac{\partial(y,z)}{\partial(u,w)}\,\frac{\partial(z,x)}{\partial(u,w)}\,\frac{\partial(x,y)}{\partial(u,w)}\right)^T. \qquad (74)$$

The typical patch has four boundary curves and $\bar{N}$ is easily computable. However, three sided (octant of a sphere) and even two-sided patches do exist, i.e., the length of a boundary curve can be zero. For these cases, the Jacobian is singular and the usual normal is not defined at the degenerate point. The coding of the computer program, that computes the coefficients in matrix array for the surface normals, recognizes the degenerate curve and performs the appropriate vector arithmetic on the two curves that actually intersect at the point. The form of this matrix array is the same as the standard four-sided patch. Figure 15 shows the surface normal vectors on a PC patch.

Planar/non-planar patch test

Sometimes it is necessary to know whether a given patch is planar or non-planar, for example, in blending operations. Rather than just "plugging in" uw values and determining whether the associated x, y, z points lie in a plane—certainly somewhat laborious and definitely not rigorous—consider the following analysis which is an extension of the technique suggested by S. L. Martin.[18]

Using the definition of a plane

$$\overline{n(0,0)}\cdot\overline{(r(u,w)-r(0,0))}=0 \qquad (75)$$

where $\overline{n(0,0)}$ is a vector in the direction of the patch normal at $(0,0)$, $\overline{r(u,w)}$ is the position vector to a general (u,w), and $\overline{r(0,0)}$ is the position vector to the point $(0,0)$. If (75)

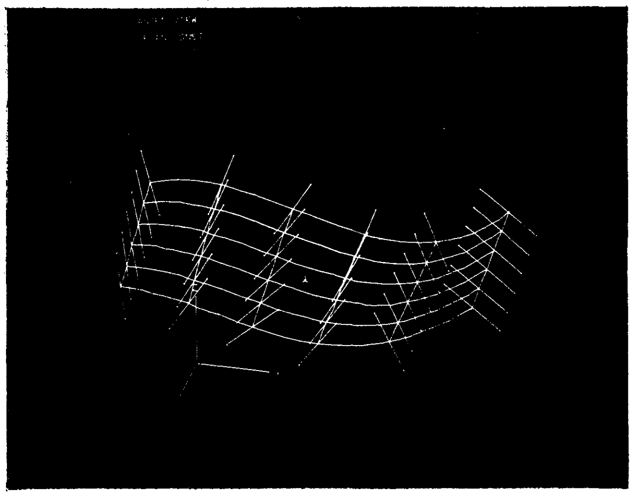


Figure 15—Surface normal vectors on a PC patch

is true for all $u, w \in [0, 1]$, then the patch is planar. Expanding the above equation

$$n_x(0, 0)x(u, w)+n_y(0, 0)y(u, w)+n_z(0, 0)z(u, w)$$
$$=n_x(0, 0)x(0, 0)+n_y(0, 0)y(0, 0)+n_z(0, 0)z(0, 0) \quad (76)$$

and note that the right hand side of (76) is a constant

$$K(0, 0)=\overline{n(0, 0)}\cdot\overline{r(0, 0)}. \quad (77)$$

Now witness

$$\int_0^1\int_0^1 \overline{n(0, 0)}\cdot\overline{r(u, w)}\, du\, dw=\int_0^1\int_0^1 K(0, 0)\, du\, dw \quad (78)$$

or

$$n_x(0, 0)\int_0^1\int_0^1 x(u, w)\, du\, dw+n_y(0, 0)\int_0^1\int_0^1 y(u, w)\, du\, dw$$
$$+n_z(0, 0)\int_0^1\int_0^1 z(u, w)\, du\, dw=K(0, 0). \quad (79)$$

It is seen that if the above integration can be performed, then the constraint conditions for planarity will be independent of uw explicitly. Recall from (63) that the general patch coordinate is

$$V(u, w)=(U)(S)(W)^T$$

which can also be written

$$V(u, w)=\sum_{i=1}^{4}\sum_{j=1}^{4} s_{ij}u^{4-i}w^{4-j}. \quad (80)$$

Hence

$$\int_0^1\int_0^1 V(u, w)\, du\, dw=\sum_{i=1}^{4}\sum_{j=1}^{4}\frac{s_{ij}}{(5-i)(5-j)} \quad (81)$$

and when this result is used in (79)

$$n_x(0, 0)\sum_{i=1}^{4}\sum_{j=1}^{4}\frac{s_{ijx}}{(5-i)(5-j)}+n_y(0, 0)\sum_{i=1}^{4}\sum_{j=1}^{4}\frac{s_{ijy}}{(5-i)(5-j)}$$
$$+n_z(0, 0)\sum_{i=1}^{4}\sum_{j=1}^{4}\frac{s_{ijz}}{(5-i)(5-j)}-K(0, 0)=0. \quad (82)$$

If the coefficients of the patch satisfy (82), then the patch is planar.

Geometric properties

The parametric form of the surface lends itself very nicely to computation of geometric properties since explicit points do not have to be computed.

Consider the determination of surface area

$$dA=\|\bar{N}(u, w)\|\, du\, dw=f_1(u, w)\, du\, dw \quad (83)$$

where $\bar{N}$ is from (73) and dA is the scalar element of area. Hence

$$A=\int_0^1\int_0^1 f_1(u, w)\, du\, dw. \quad (84)$$

Using Gauss quadrature

$$A=\sum_{j=1}^{N}\sum_{i=1}^{N} g_i h_j f_1(u_j, w_i) \quad (85)$$

where g_i and h_j are the weight values associated with the specific N-point formula.

For the volume of a closed region, it can be shown[19] that

$$dv=\tfrac{1}{3}[\bar{r}(u, w)\cdot\bar{N}(u, w)]\, du\, dw=f_2(u, w)\, du\, dw \quad (86)$$

where $\bar{r}$ and $\bar{N}$ are the position vector and normal vector, respectively, and dv is the scalar element of volume. There-

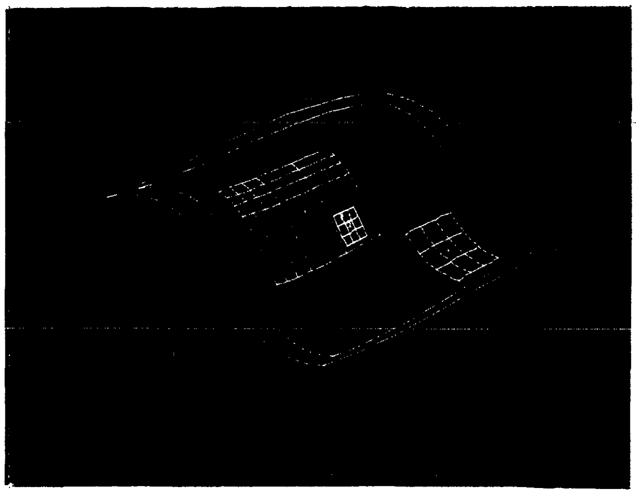


Figure 16—PC patch subdivision or extension

fore,

$$v=\int_0^1\int_0^1 f_2(u,w)\,du\,dw \tag{87}$$

and again quadrature can be used as in (85).

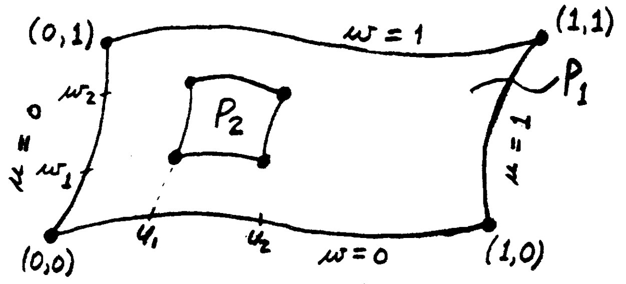
As for PC arc length, $N=9$ is used to compute both surface area and volume. Other desirable properties such as moments of inertia, center-of-gravity, and principal axes information are obtainable.

Creating a patch from a given patch

Three techniques will now be presented whereby a new patch can be designed which has similar geometric properties to a given patch.

Subdivide/extend

Consider the following sketch.



It is desired to create patch P_2 from P_1 with the indicated new corner points determined by u_1, u_2, w_1, and w_2. The (B) matrix for P_2 is obtained as follows: For position and parametric slope data, the PC segmentation principles of (35) are employed, e.g.,

$$(V(0,0))_{P_2}=(V(u_1,w_1))_{P_1} \tag{88}$$

and

$$(V(0,0)_u)_{P_2}=(u_2-u_1)(V(u_1,w_1)_u)_{P_1}. \tag{89}$$

The corner cross derivatives for P_2 are obtained by evaluating the partial derivatives of P_1 at the given subdivide points, e.g.,

$$(V(1,1)_{uw})_{P_2}=\left[\left(\frac{\partial^2 V(u,w)}{\partial u\,\partial w}\right)_{P_1}\right]_{u=u_2,w=w_2}. \tag{90}$$

Therefore, the entire geometric form for P_2 can easily be obtained. As a point of interest, the patch can be extended beyond its original borders; however, this is not too advisable since extrapolation is dangerous (at least, large extrapolation). Figure 16 shows several examples of the subdivide/extend capability.

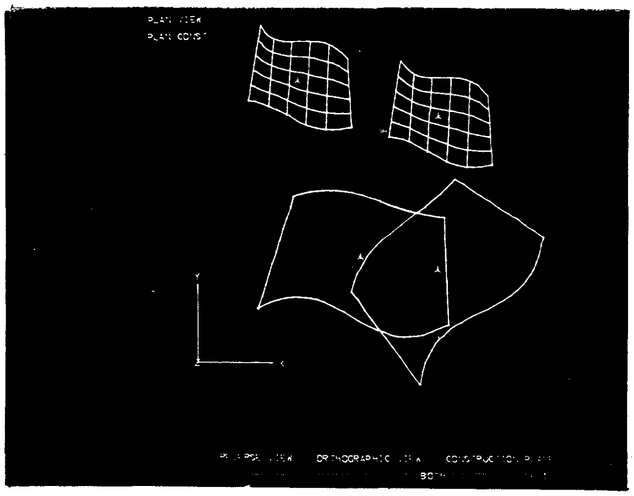


Figure 18—PC patch translation and rotation

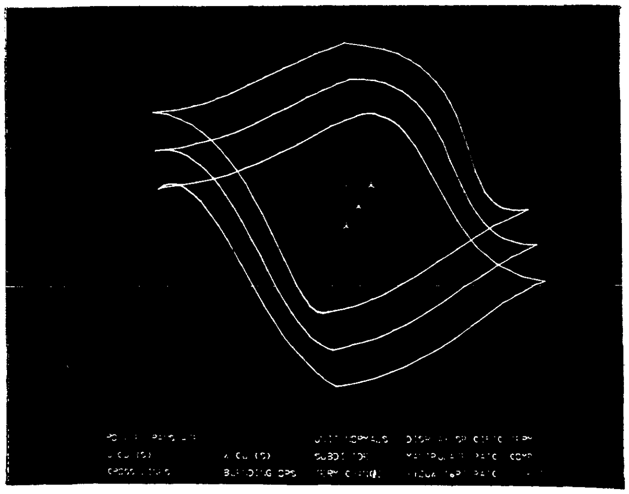
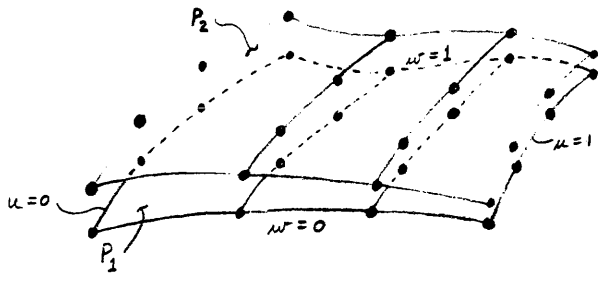


Figure 17—PC patch offsetting

Offset a patch

Given a patch P_1, create an offset patch P_2 governed by some rule. See the following sketch.



Select 16 convenient points, e.g.,

$$(u_i,w_i) \qquad u,w=0,\tfrac{1}{3},\tfrac{2}{3},1 \tag{91}$$

then the offset points can be generated by

$$(V(u,w))_{P_2}=(V(u,w))_{P_1}+(n_V(u,w))_{P_1}L_V(u,w) \quad (92)$$

where n_V is the unit normal vector component at u_i, w_i on P_1 and $L_V(u,w)$ is the law or rule which will produce the offset points. Therefore, parallel or tapered thickness surfaces can be created. Of course, P_2 is truly parallel at 16 points only. Figure (17) shows the offset capability for a constant value of L_v.

Translation, rotation, and scaling

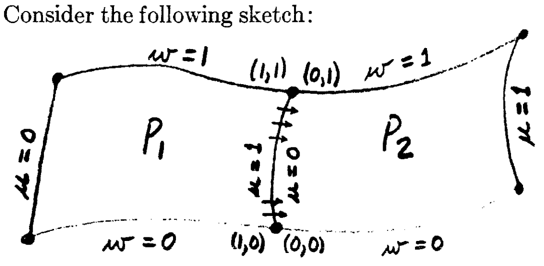
Transformation operations on the PC patch are very similar to those already discussed for the PC curve in equations (37), (38), and (39). Again the operations are performed on equations, not just data points. Suffice it to say that for translation in algebraic form, ΔV is added to the s_{44} element of the (S) matrix; while in geometric form, ΔV is added to each element of the P partition of the (B) matrix in (67). Rotation and scaling are exactly similar. Figure 18 shows the transformation operations on a PC patch.

Blending patches

Three specific cases will be discussed. These are: (1) blending at a common border, (2) blending between two non-adjacent patches, and (3) blending a given patch to the borders of another patch.

Blending at a common border

Consider the following sketch:



The two patches are to be blended such that $C^{(0)}$ and $C^{(1)}$ continuity exist across the common boundary.

It is now necessary to discuss a most important item: the slopes across the boundary curve of a PC patch. Indeed, a powerful feature of the patch is the fact that the slopes across a boundary curve can be expressed as an actual equation by mere inspection of the geometric form in (56). The equation of the parametric slopes in the u direction along the $u=1$ curve of P_1 is

$$\left[\frac{\partial V(u,w)}{\partial u}\right]_{u=1}\equiv(1,w)_u$$

$$=F_1(w)V_{1ou}+F_2(w)V_{11u}$$

$$+F_3(w)V_{1ouw}+F_4(w)V_{11uw}. \quad (93)$$

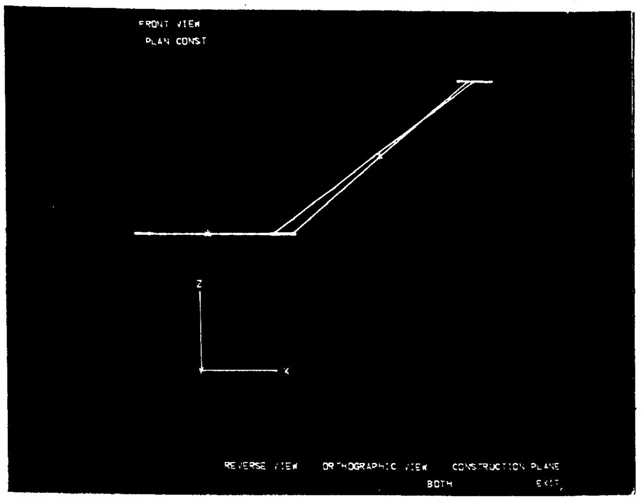


Figure 19—Blending PC patches at a common border, slope discontinuity prior to blending

Note that the cross-derivatives affect the slopes all along the boundary except at $w=0$ and $w=1$ where F_3 and F_4 are both zero.

As pointed out in (24), multiplying all the parametric slopes by a constant k just scales up or down on the real space slope vector, i.e., direction cosine data are not changed. Hence, with respect to the form of the (B) matrix, the blending operations for the sample sketch are

$$(u=0,\text{ row }1)_{P_2}=(u=1,\text{ row }2)_{P_1} \qquad C^{(0)}\text{ continuity} \quad (94)$$

and

$$(u=0,\text{ row }3)_{P_2}=k(u=1,\text{ row }4)_{P_1}. \qquad C^{(1)}\text{ continuity} \quad (95)$$

Therefore, adjacent patches have position and slope continuity only if common position rows (or columns) are identical and if common slope rows (or columns) are multi-

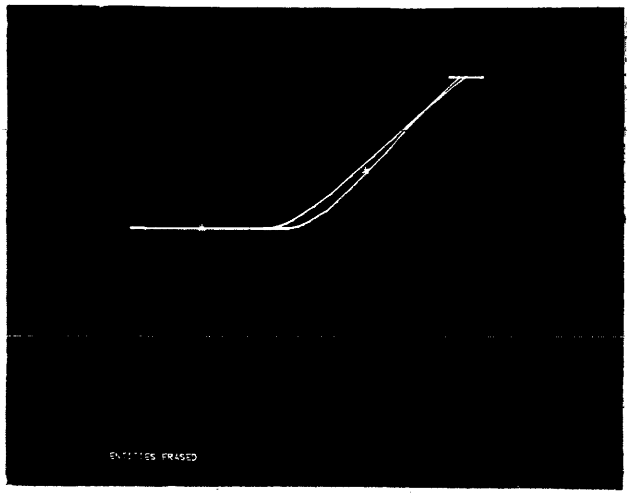


Figure 20—Blending PC patches at a common border, slope continuity after blending

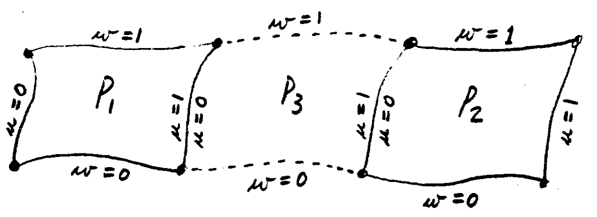
ples of each other. Graphically, the elements of interest in the (B) matrices for both patches of the sample are

$$\begin{array}{c|cccc|cccc|}
 & \multicolumn{4}{c|}{P_1} & \multicolumn{4}{c|}{P_2} \\
\hline
 & & & & & b_{21} & b_{22} & b_{23} & b_{24} \\
 & b_{21} & b_{22} & b_{23} & b_{24} & & & & \\
 & & & & & kb_{41} & kb_{42} & kb_{43} & kb_{44} \\
 & b_{41} & b_{42} & b_{43} & b_{44} & & & & \\
\hline
\end{array}$$

Figures 19, 20, and 21 show the blending operation sequence for adjacent patches. Figure (19) shows the obvious slope discontinuity at the common border prior to blending; Figure (20) shows the slope continuity after blending; and Figure (21) depicts the blended patches in another view. The bottom patch lies completely in the xy plane while the top patch is truly three-dimensional, i.e., has depth (z) values.

Blending between two non-adjacent patches

This case is characterized by the following sketch:



Patch P_3 is to be created such that it manifests $C^{(0)}$ and $C^{(1)}$ continuity with patches P_1 and P_2 at the respective common borders. The same technique is applied as in section 5.8.1,

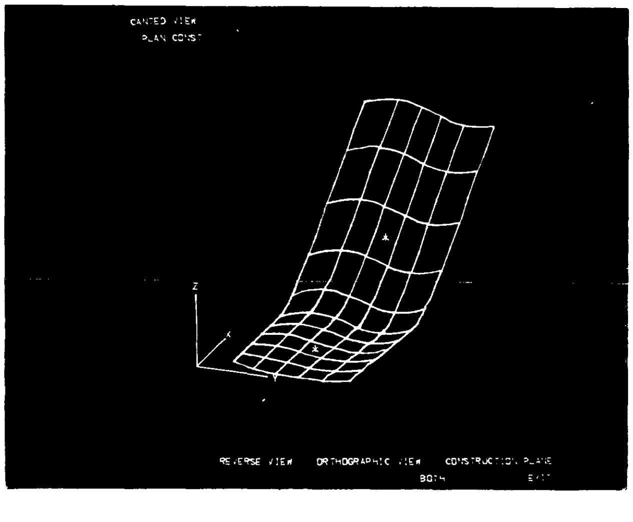


Figure 21—Blending PC patches at a common border, bottom patch in X-Y plane

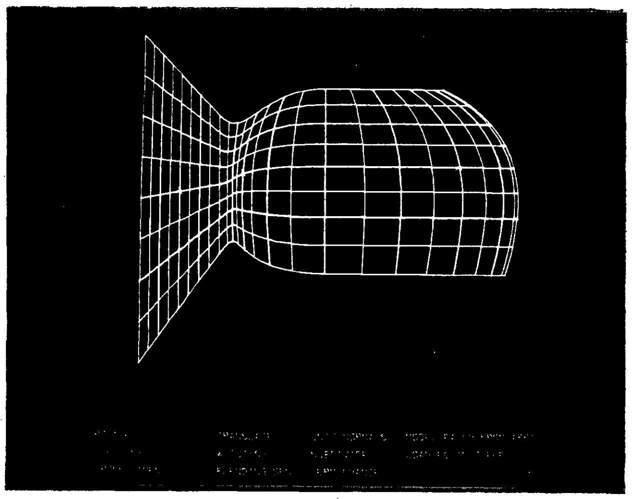


Figure 22—Blending a Pc patch between two non-adjacent patches, plan view

that is

$$(u=0, \text{row } 1)_{P_3} = (u=1, \text{row } 2)_{P_1} \quad (96)$$

$$(u=1, \text{row } 2)_{P_3} = (u=0, \text{row } 1)_{P_2} \quad (97)$$

$$(u=0, \text{row } 3)_{P_3} = k_1(u=1, \text{row } 4)_{P_1} \quad (98)$$

$$(u=1, \text{row } 4)_{P_3} = k_2(u=0, \text{row } 3)_{P_2} \quad (99)$$

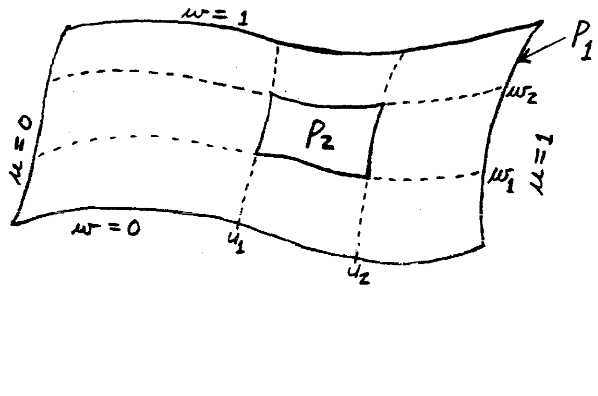
Equations (96) and (97) guarantee $C^{(0)}$ continuity while (98) and (99) guarantee $C^{(1)}$ continuity. Graphically, the elements of interest in the boundary matrix are

$$\begin{array}{c|cccc|cccc|cccc|}
 & \multicolumn{4}{c|}{P_1} & \multicolumn{4}{c|}{P_3} & \multicolumn{4}{c|}{P_2} \\
\hline
 & & & & & b_{21} & b_{22} & b_{23} & b_{24} & B_{11} & B_{12} & B_{13} & B_{14} \\
 & b_{21} & b_{22} & b_{23} & b_{24} & B_{11} & B_{12} & B_{13} & B_{14} & & & & \\
 & & & & & k_1b_{41} & k_1b_{42} & k_1b_{43} & k_1b_{44} & B_{31} & B_{32} & B_{33} & B_{34} \\
 & b_{41} & b_{42} & b_{43} & b_{44} & k_2B_{31} & k_2B_{32} & k_2B_{33} & k_2B_{34} & & & & \\
\hline
\end{array}$$

Figures 22 and 23 show a patch blended between two non-adjacent patches. The patch on the left is ruled and in the xy plane while the right patch is a 90 degree surface of revolution about the y axis.

Blending a patch to the borders of another patch

Consider the following sketch:



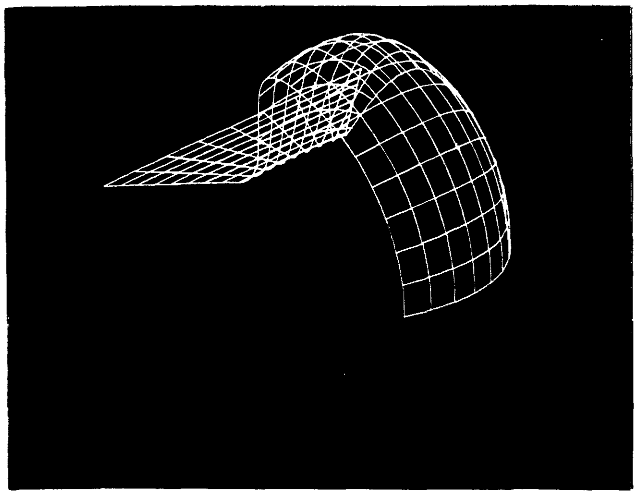
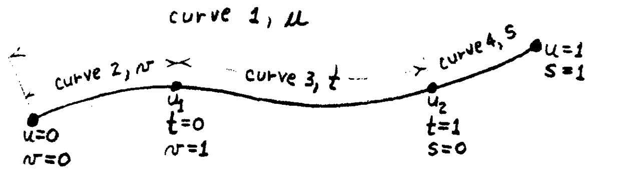


Figure 23—Blending a PC patch between two non-adjacent patches, trimetric view

The objective is to blend P_2 into the borders of P_1 and this will be accomplished by creating eight blending patches indicated by the dashed lines. Again, the key to this operation is the relationship between the parametric tangents when a PC is segmented. From the following sketch



and using (34) of section 4.6, it can be shown that

$$\left(\frac{dV}{dt}\right)_{t=0} = \left(\frac{u_2-u_1}{u_1}\right)\left(\frac{dV}{dv}\right)_{v=1} \qquad (100)$$

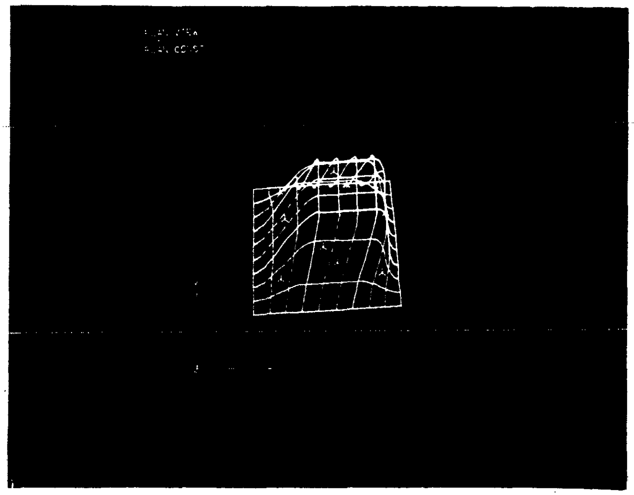
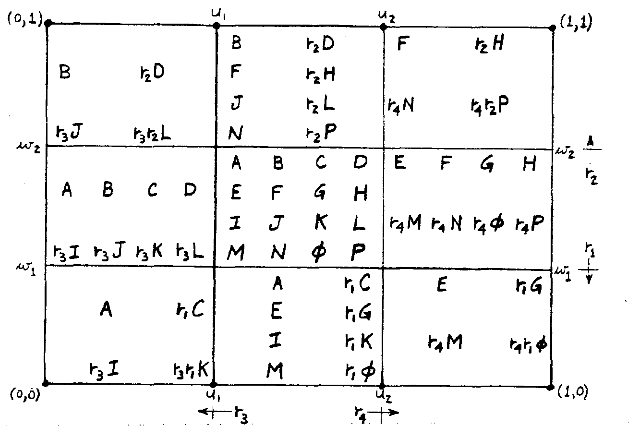


Figure 24—Blending a PC patch to the borders of another patch

and

$$\left(\frac{dV}{dt}\right)_{t=1} = \left(\frac{u_2-u_1}{1-u_2}\right)\left(\frac{dV}{ds}\right)_{s=0}. \qquad (101)$$

Now, applying the ideas of the previous sections, it can be shown that the elements of interest in the (B) matrices are



This format conveniently combines matrix information with uw values. The blank entries are dictated by surrounding patches. A. L. Eshleman of Douglas Aircraft, who originated this technique, calls these blank spaces "freedom elements" since they can be arbitrary if there are no surrounding patches. The ratio values

$$r_1 = \frac{w_1}{w_2-w_1}$$

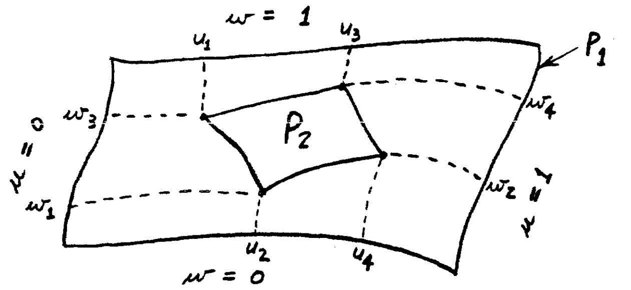
$$r_2 = \frac{1-w_2}{w_2-w_1}$$

$$r_3 = \frac{u_1}{u_2-u_1}$$

$$r_4 = \frac{1-u_2}{u_2-u_1} \qquad (102)$$

guarantee $C^{(1)}$ continuity across the boundaries. Figure 24 shows a surface generated by the "blend eight" operation.

The general "blend eight" case can be pictured as

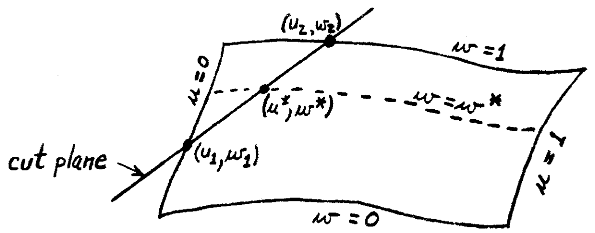


This case has been implemented and it involves eight ratios which of course reduce to four for the special case described previously.

If the patch to be blended, P_2, is planar, then both patches can be rotated such that P_2 is in a principal plane, e.g., xy plane; this effectively reduces the computations involved. After blending, the patches are rotated back.

Plane and patch intersection

Consider the following sketch:



In order to generate "cut points" to be fitted later by PC curves, witness

$$\Delta w = w_2 - w_1 \qquad w_2 > w_1 \tag{103}$$

where w_1 and w_2 are obtained by intersecting a boundary curve or curves by the plane. One of the values for w_1 or w_2 is not necessarily 0 since a plane could intersect only one boundary curve. Let

$$p = \frac{w - w_1}{w_2 - w_1} \qquad 0 \leq p \leq 1 \tag{104}$$

hence

$$w = w_1 + p\Delta w. \tag{105}$$

For $p = p^*$,

$$w^* = w_1 + p^*\Delta w \tag{106}$$

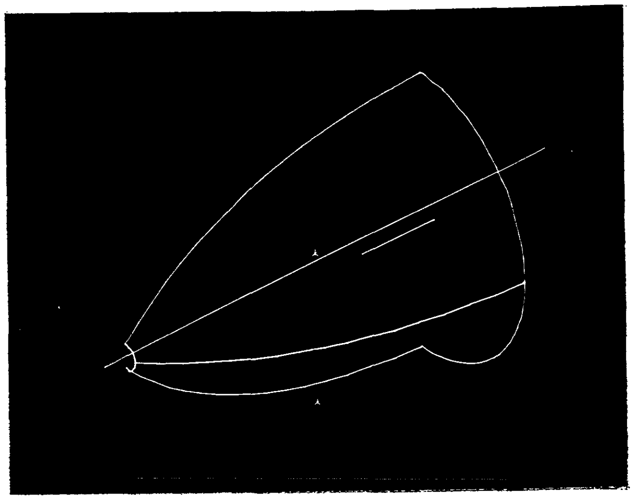


Figure 26—Surface modification to include straight hinge line-problem statement

and $u = u^*$ along the line $w = w^*$ is produced by cutting w^* with the plane. Hence, the point on the patch surface $V(u^*, w^*)$ is easily computed. The points produced in this fashion are then fitted with a PC curve(s) to a given tolerance. The major point is that actual continuous curves are produced, not a string of discrete data points. Figure 25 shows the intersection of a compound curvature patch by a family of planes. The solid lines are the resultant intersection curves while the dashed lines are constant uw lines used for display.

ENGINEERING DESIGN APPLICATIONS

The following typical design problems and their solutions validate the claim that the PC patch is a realistic, efficient, and accurate device for CAD interactive graphics.

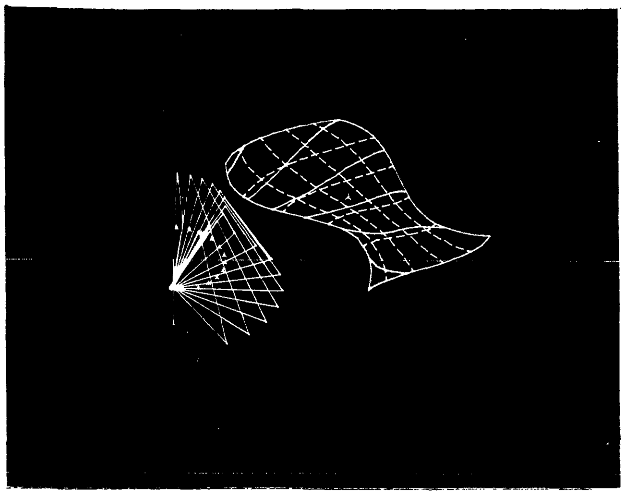


Figure 25—Intersecting a PC patch with a plane

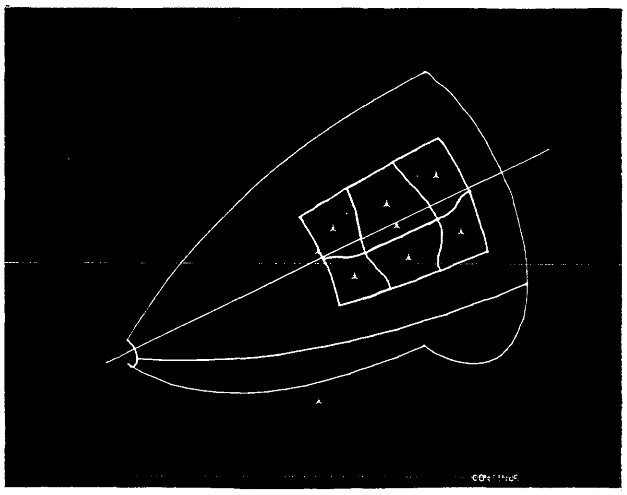


Figure 27—Surface modification—Solution showing blending patches

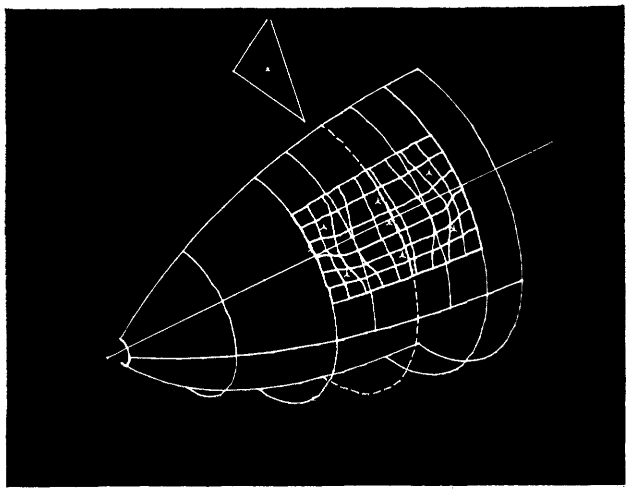


Figure 28—Surface modification—Plane cut through hinge line

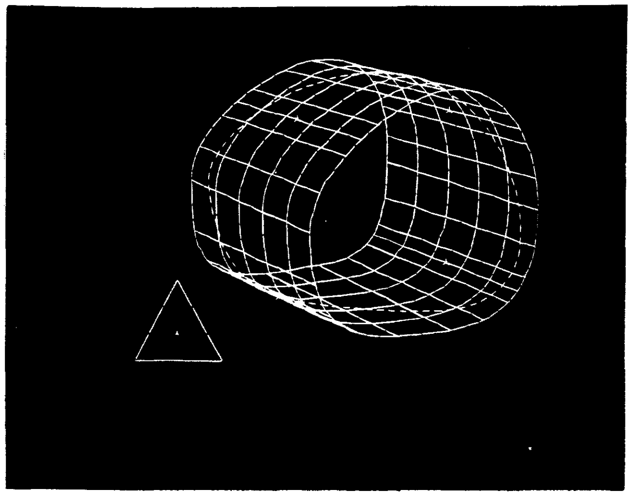


Figure 30—Engine inlet duct cut by a plane

Surface modification to incorporate a hinge line

The problem statement is depicted in Figure 26 where the two end points of the hinge line lie on the surface of the body of revolution (e.g., radome or nose cone). The local neighborhood of the surface near the hinge is to be modified. The solution is shown in Figure 27 where the blending patches are created by using the principles previously described. Figure 28 shows a plane cut through the hinge line while Figure 29 shows the edge view of the cut plane.

Engine inlet duct

The problem is to design an inlet duct such that plane cuts normal to the air flow center-of-gravity satisfy aerodynamic and propulsion requirements. It should be noted that typical duct contours possess points of inflection. Of course, this presents no difficulty for the PC. Figure 30 shows the duct cut by a plane and the resultant closed and continuous PC curve lying on the surface. Figure 31 shows the edge view of the plane.

Creation of bolt holes for access door

The problem statement is to construct two bolt holes at a specified location on a surface. The solution employs PC surface subdivision and patch revision through blending. Figure 32 shows the "bossed areas" blended into the original surface.

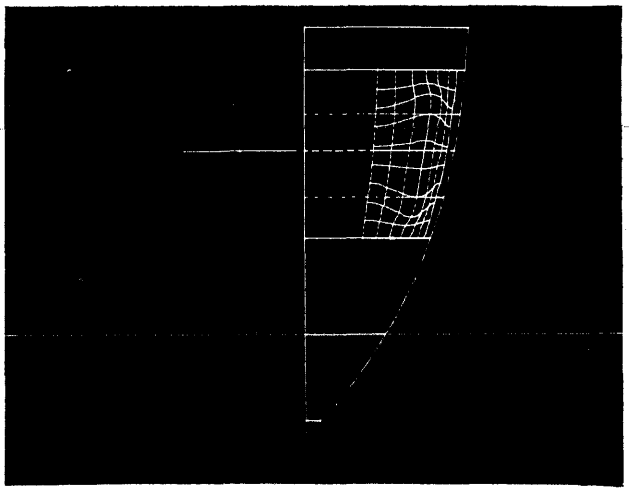


Figure 29—Surface modification—Edge view of cut plane

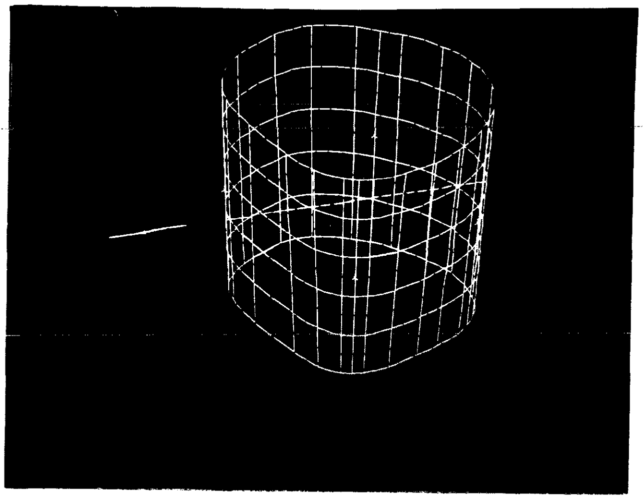


Figure 31—Engine inlet duct—Edge view of cut plane

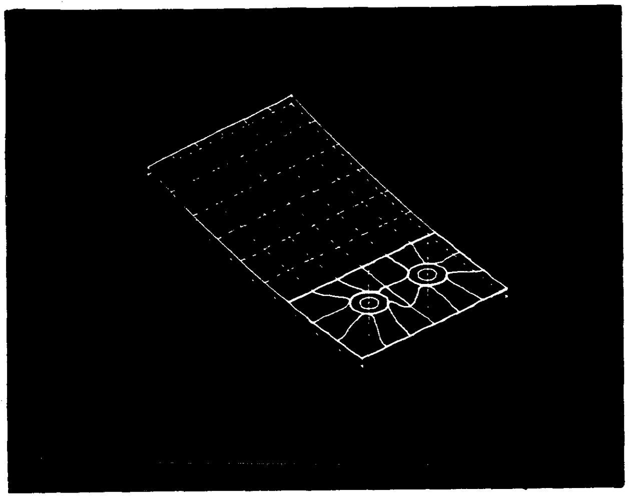


Figure 32—Creation of bolt holes for access door

Thermal insulation tile for re-entry vehicle

The problem statement is to create a parallel or thick surface. The solution involves specifying the desired thickness and employing the offset capability to the given patch. Figure 33 shows the designed tile.

Stress analysis

The success of the discrete or finite element method in solving practical structural mechanics problems has led to its increased use in other disciplines of engineering mechanics. In large part, the success of the method has been the result of its ability to represent the geometric irregularities so often present in actual hardware. Ironically, most efforts to improve the method have ignored the representation of the initial geometry and have focused on improving the representation of the deformed geometry (analysis variables). This has led to many new plate and shell discrete elements whose accuracy decreases sharply for structures with initially curved boundaries. The number of straight-sided quadrilateral and/or triangular elements needed to accurately solve an elliptic-plate problem is significantly higher than that required for a square plate. This behavior is directly related to the poor approximation of the curved plate boundary with straight-sided discrete elements. Reference 20 provides several excellent examples of this behavior associated with a higher order triangular discrete element. To avoid the situation, it is enough to return to the basic strength of the discrete-element method and improve the representation of the initial geometry. Parametric discrete elements, specifically PC representation, provide this improved modeling for both the deformed and undeformed geometry.

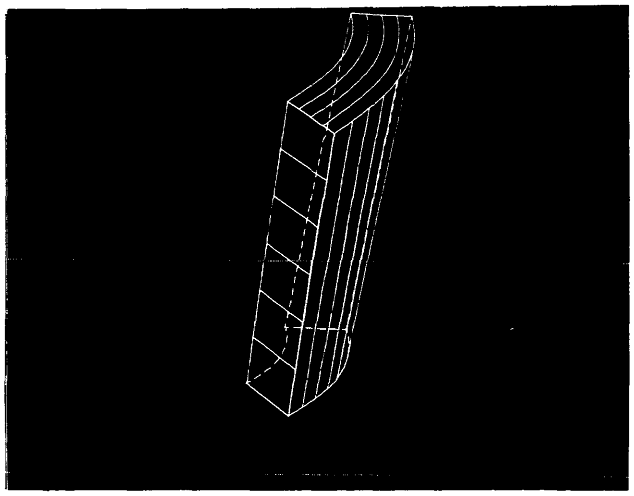


Figure 33—Thermal insulation tile for reentry vehicle

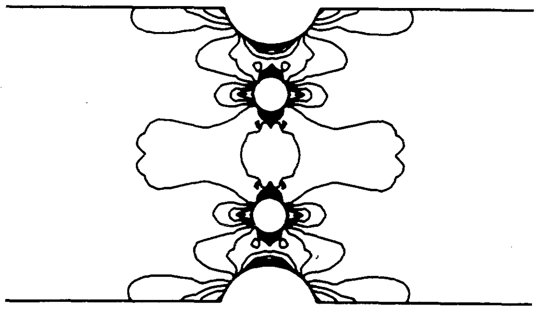


Figure 34—Stress analysis of specimen, plan view isogram of principal strain difference

Figure 34 shows a pseudo-elastic picture obtained by computing the principal strain differences, fitting them with PC surface patches over each element and slicing these surfaces with equally spaced planes. The spacing of the cutting planes is proportional to the fringe constant of the photoelastic material (Homalite 100), and hence they should produce the same "fringe" pattern as the photoelastic test

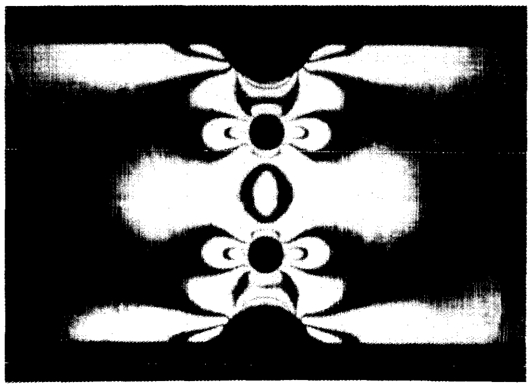


Figure 35—Stress analysis of specimen, photoelastic test results

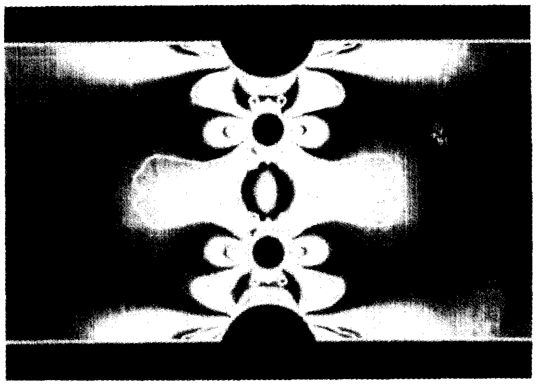


Figure 36—Stress analysis of specimen, overlay of photoelastic results and parametric solution

results depicted in Figure 35. How close the parametric discrete element solution comes to this condition can be seen in the overlay shown in Figure 36. These rather remarkable figures were obtained through the courtesy of E. L. Stanton.[21] Although these results were not obtained interactively, an ICG implementation is obviously desirable since a change in the patches that define the plate geometry automatically would create a new discrete-element model for analysis. Also, much of the existing software developed for surface representation can be used to automate the geometry data for discrete element analysis and to plot the stress and displacement results.

CONCLUSIONS

This paper has discussed the implementation of the PC curve and surface patch from the following aspects: the desirability of the parametric representation, the computer system environment, the mathematical formulations, and ICG engineering design problems.

Some of the features of this design device which make it a very attractive candidate for computer-aided design are:

(1) All curves and surfaces are used in a single form. (The PC curves are a subset of PC surfaces).
(2) Planar or twisted data can easily be handled.
(3) Infinite slopes require no special handling.
(4) Rotation, translation, and scaling are done on equations.
(5) Arc length, surface area, volume, etc. are computed directly without computing explicit points.
(6) Coordinates of end points of curves and corner points of patches are used directly as coefficients of curves in the PC equation thus providing an aid to the identification of surface patches and connectivity to other patches.
(7) Cross derivative coefficients alter the shape of the surface without affecting the boundary PC curves.
(8) Patches are bounded by four space curves. However, curves may be of zero length, i.e., 2-sided (not too useful) or 3-sided patches are possible.
(9) Two or more patches can be matched identically along one boundary of an adjacent patch.
(10) There exists a one-to-one mapping for analysis results; i.e., stress, strain, temperature, pressure, etc., distributions can be obtained.

Parametric cubic curves and patches form a realistic, flexible surface system suitable for describing most engineering shapes. It should be emphasized that the designer's geometric inputs at the CRT are transformed into actual surface mathematics in such a way that the mathematics are completely hidden, unless requested otherwise, from him. Hence, the experienced designer is free to concentrate on attaining the solution to his defined problem or indeed to create the definition of his problem.

ACKNOWLEDGMENTS

The author expresses his gratitude to the following persons: Mr. R. Rolph for his vast experience in lofting techniques and programming which facilitated the implementation of the PC in the Design/Drafting package; Mr. G. Gilsinn for many helpful discussions and programming relative to the PC; Mr. A. Eshleman and Mr. H. Meriwether for introducing him to the PC and providing the batch programs which formed the kernel of ICG surface development in St. Louis; and Mr. J. J. Lavick, and Mr. B. K. Winters for their managerial insight into seeing potential value in a design technique which merited investigation. Lastly, appreciation is expressed to the entire Computer-Aided Design Drafting team in St. Louis, both computer programmers and engineers, for excellent achievement over a period of several years in providing an interactive engineering design capability.

REFERENCES

1. Coons, S. A., *Surfaces for Computer-Aided Design of Space Figures*, M.I.T. ESL Memorandum 9442-M-139, January 1964.
2. Coons, S. A., *Surfaces for Computer-Aided Design of Space Forms*, Project MAC, Massachusetts Institute of Technology 1967.
3. Ferguson, J. C., *Multi-variable Curve Interpolation*, The Boeing Company, Document D2-22504, July 1963.
4. Forrest, A. R., *Curves and Surfaces for Computer-Aided Design*, Doctoral thesis, University of Cambridge, 1968.
5. Eshleman, A. L. and H. D. Meriwether, *Graphic Applications to Aerospace Structural Design Problems*, SHARE Annual Design Automation Workshop, Los Angeles, 1967.
6. Lavick, J. J., *Design Philosophies for a Man-Machine Engineering Environment*, McDonnell Douglas Automation Report No. 6045, 1967.
7. Lavick, J. J., *Computer-Aided Design at McDonnell Douglas*, International Symposium (CG 70), Brunel University, Uxbridge, England, April 1970.
8. Lavick, J. J. and G. L. Martin, *Modern Techniques in Design*, 1972 CAD/CAM Conference, Society of Manufacturing Engineers, Atlanta, Georgia, February 1972.

9. Newell, A., *A General Discussion of the Use of Conic Equations to Define Curved Surfaces*, The Boeing Company, Document D2-4398, March 1960.
10. Shelley, J. H., *The Development of Curved Surfaces for Aero-Design*, Gloster Aircraft Co., Ltd., 1947.
11. Ahlberg, J. H., E. N. Nilson and J. L. Walsh, *The Theory of Splines and their Application*, Academic Press, 1967.
12. Acton, F., *Numerical Methods that Work*, Harper and Row, New York, 1970.
13. Brand, L., *Vector and Tensor Analysis*, John Wiley and Sons, New York, 1947.
14. Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956.
15. *Handbook of Mathematical Formulas*, U. S. Government Printing Office, Washington, D. C., 1970.
16. Herzog, B. and G. Valle, *Interactive Control of Surface Patches*, International Symposium (CG 70), Brunel University, Uxbridge, England, April, 1970.
17. Pennington, R. H., *Introductory Computer Methods and Numerical Analysis*, Macmillan Company, Toronto, 1970.
18. Martin, S. L., Informal notes, presently at Stanford University, formerly with Douglas Aircraft, Long Beach, California.
19. Buck, R. C., *Advanced Calculus*, McGraw-Hill, New York, 1956.
20. Chernuka, M. W., G. R. Cowper, G. M. Lindberg and M. D. Olson, *Application of the High Precision Triangular Plant-Bending Element to Problems with Curved Boundaries*, NAE Aeronautical Reports LR-529, National Research Council of Canada, October 1969.
21. Stanton, E. L. and E. L. Palacol, "Anisotropic Parametric Plate Discrete Elements," *International Journal for Numerical Methods in Engineering*, Volume 6, 1973.

Twinkle Box—A three-dimensional computer input device

by ROBERT P. BURTON

Bell Telephone Laboratories
Holmdel, New Jersey

and

IVAN E. SUTHERLAND

Evans and Sutherland Computer Corporation
Salt Lake City, Utah

INTRODUCTION

During the past fifteen years, use of two-dimensional computer input/output devices has become commonplace. Since the earliest uses of the light pen for target identification in air defense systems it has been obvious that two-dimensional input would be interesting and useful. A large number of two-dimensional tablets and digitizers have been developed and have come into quite effective use. These devices have made use of mechanical, electrical, magnetic, optical, and acoustic phenomena. (See bibliographical references.)

More recently, the use of three-dimensional computer output devices has become prominent. It seemed obvious at first that corresponding three-dimensional computer input devices might be interesting and useful, but there has been no corresponding development of these devices. Rather, there has been a series of laboratory developments each with limited utility. A three-dimensional version of the Science Accessories Corporation acoustic tablet is the only commercially available three-dimensional input device with which we are acquainted.[9] Why has there been no prominent development of three-dimensional input devices? Aside from the obvious reason that three-dimensional graphics is used less than two-dimensional graphics, there are three reasons why three-dimensional input is not more widespread. First, early experiments with three-dimensional graphics have shown that people are not very good at drawing in space without the support of a writing surface. The years of training that grade school children go through in learning to write do not facilitate three-dimensional input. Second, the measurement of three-dimensional positions is substantially more difficult than that of two-dimensional positions. The measuring signals must travel through free space; the measuring device cannot be embedded in a surface. Finally, the coordinate conversion required to reduce the measurements actually taken to Cartesian coordinates is generally much more complex for a three-dimensional device.

This paper describes another laboratory development of limited utility.[2] We think this new device is interesting because, unlike previous devices which have measured the position of only a single point, it measures the positions of many three-dimensional points in such rapid succession that they appear to be measured simultaneously. We also feel that our novel approach to the coordinate conversion problem may be useful to others.

BACKGROUND

The earliest three-dimensional computer input device with which we are familiar is the Lincoln Wand, demonstrated by Lawrence G. Roberts in 1963.[8] Roberts' device used an ultrasonic signal and four microphones mounted at the corners of a rectangle. The path lengths from the point source of sound to each microphone were determined by the arrival times of the pulse at each microphone. As Roberts showed in his paper, the rectangular arrangement of microphones made the coordinate conversion problem fairly easy.

Concurrent with Roberts' work, Jack Raffel at Lincoln Laboratory proposed a photosensing device. Raffel's idea (never published) was to measure the ratio of the illumination falling on two photocells placed at right angles to one another. This ratio is related to the angle of arrival of the light. The position of the light could be determined from three such measurements. As will be seen later in this paper, the coordinate conversion involved would not have been particularly difficult if handled by matrix methods. However, these methods were not available to Raffel at the time.

An activity at M.I.T.[10] for measuring three-dimensional position used the critical angle of acoustic radiation into three mutually perpendicular solid rods to obtain a measurement directly in the Cartesian coordinate system defined by the rods. The idea was that the difference of arrival times at the ends of each rod locates the pulse source on a plane perpendicular to the rod and at a distance from its center proportional to the difference measured.

More recently, A. Michael Noll has developed an electromechanical three-dimensional input device in conjunction

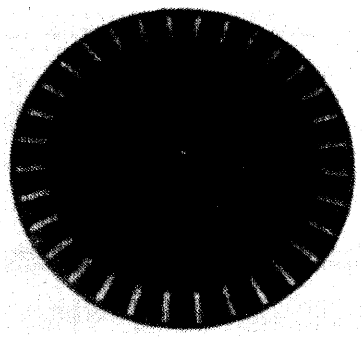


Figure 1—Disk

with a stereo display.[6] This device permits direct input in Cartesian coordinates and apparently works quite well. A group under Frederick Brooks at the University of North Carolina has been working on a combined three-dimensional input/output device. The device receives input from remotely controlled mechanical limbs, such as those used in handling radioactive materials.[3] The device can be moved by man and by the computer, serving not only for three-dimensional input, but also as a force display.

The most successful device now available is a three-dimensional adaptation of the Science Accessories Corporation acoustic tablet.[9] This device measures the time required for sound to travel from a small spark source to each of three mutually orthogonal linear microphones. The arrival times indicate the position of the spark gap. However, the Science Accessories device has several weaknesses due primarily to the slow speed with which sound travels in air. Sampling is limited to about one hundred measurements per second for a reasonable working volume. Accuracy is limited to one part in five hundred due to the movement of air even in a quiet room. Finally, any object intervening between the sound source and a microphone will destroy the measurement.

In addition to some difficulties in signal-to-noise ratio, accuracy, reliability, utility, etc., each of these devices measures the position of only a single point which may be moved. Our own interest in three-dimensional input developed from use of the head-mounted display[12] for which the position and orientation of the user's head must be sensed. For this purpose, the positions in space of at least three points must be measured simultaneously. Our early efforts to make these measurements acoustically (reported in Reference 12) were never satisfactory. Instead, a mechanically coupled head-position sensor has been used. The Twinkle Box, with three lights attached to a cap, could replace the bulky, mechanical headgear. Other lights could be attached to the fingertips or body to allow the user to interact with objects viewed through the head-mounted display.

The ability to sense the positions of many points in space provides for a new kind of three-dimensional input. Rather than drawing with the point of a three-dimensional pencil, a user might make broad gestures using his fingers separately. He might grasp objects to move them, indicate sizes by gesturing with his two hands, or otherwise make use of the many three-dimensional motions with which humans (particularly Frenchmen and Italians) are said to communicate. Possible use in animation comes to mind as a result of the growing capability of computers to provide realistic perspective pictures. The ability to effectively measure real body motions with a device such as the Twinkle Box should materially aid in defining the kinds of (realistic) motions which could be imparted to the animated objects or characters. We are hopeful that the ability to measure many points in space will overcome the well-known inability of people to draw in three dimensions with a free stylus.

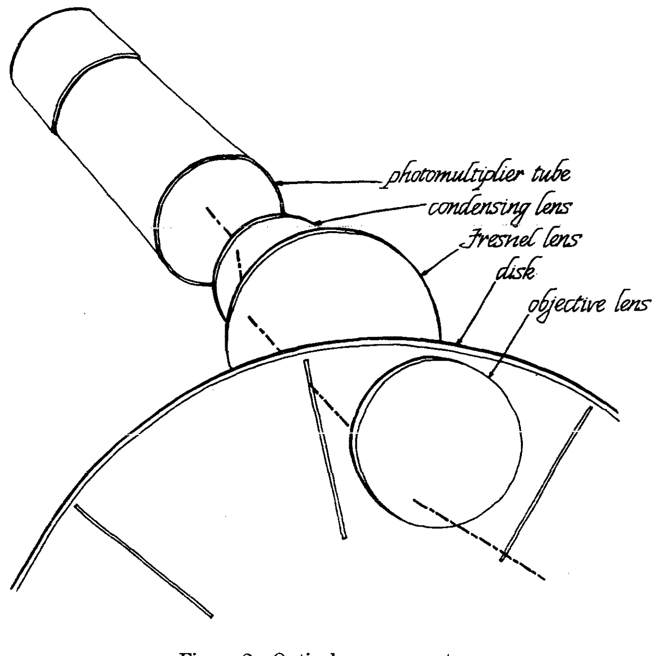


Figure 2—Optical arrangement

DETECTORS

Most people who see the Twinkle Box immediately ask why television camera technology has not been used. The

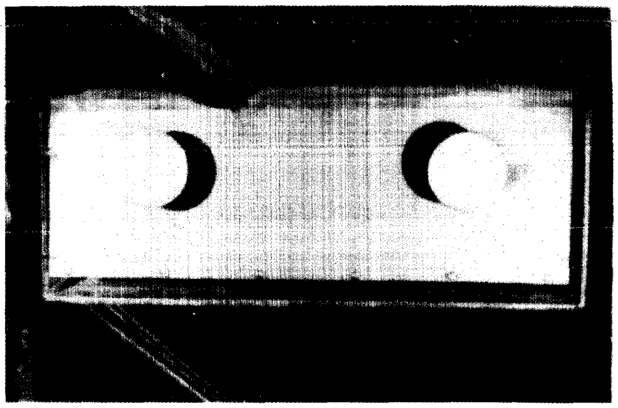


Figure 3—Detector-pair and housing

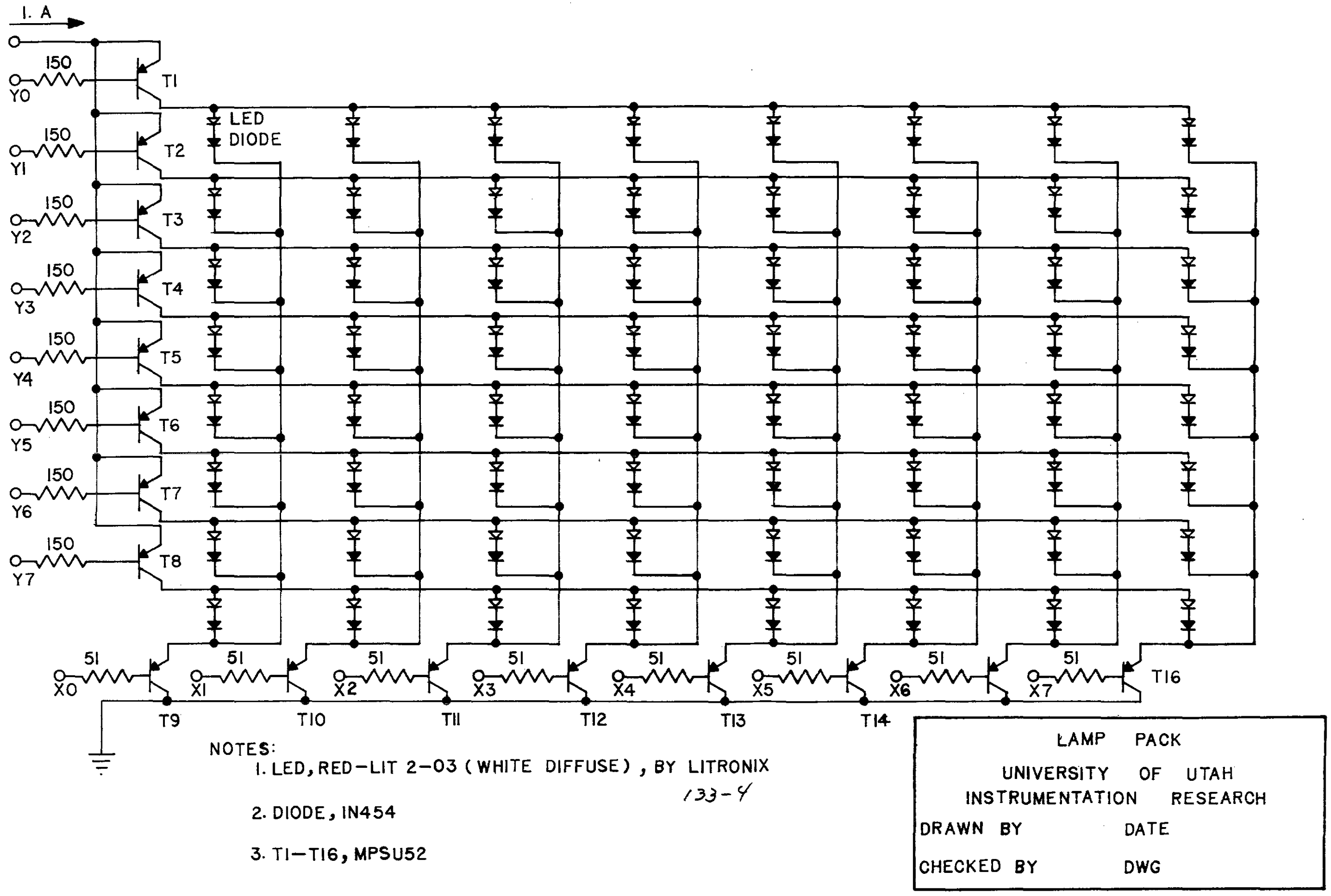


Figure 4—Diode matrix

choice of one-dimensional scanning rather than two-dimensional television type scanning is, in fact, the principal idea in the Twinkle Box design. With similar video bandwidth, hundreds of one-dimensional scans can be made in the time required for a single two-dimensional scan. The higher scanning rate makes it possible to distinguish among many light sources by turning on only one light during any one scan. Because only a single light is on at any one time, one-dimensional scans from at least three locations provide an unambiguous measure of position. To measure the positions of an equivalent number of lights using two-dimensional scanning would require a very complicated program which could match up the individual lights seen in one TV image with those seen in another image, since many lights would appear simultaneously in each image.

The Twinkle Box scanners are mechanical. Each detector-pair uses a 22-inch diameter disk with 32 radial slits cut near the edge (Figure 1). The disk rotates at 3500 rpm to provide for 1900 scans each second. Four detector-pair units are mounted in the four upper corners of a room to provide for full coverage of the room. An improved design would use some kind of electronic scanning, but mechanical scanning is sufficient to demonstrate capability.

A wide angle lens in front of the scanning disk forms a two-dimensional image of the room in the plane of the disk. When a single light is activated, this image is a single point of light which can pass through the disk only when a slit is properly positioned. Behind the disk, a Fresnel lens and a condensing lens gather light which has penetrated the slit and direct it into a photomultiplier. This optical arrangement is shown in Figure 2.

For a given orientation of the disk, the photomultiplier is sensitive to any light which lies in a plane defined by a particular slit and the center of the objective lens. As the disk rotates, the plane sweeps through the working volume of the detector. The coefficients of the plane equation are determined by the position of the slit which is determined by the time at which a pulse of light is sensed. The times at which each of three detectors sees a particular light determine three plane equations whose simultaneous solution is the position of the light.

Because the plane of sensitivity is determined by the

Figure 5—Lamp pack and user

position of a radial slit and the center of the objective lens, one might think that accurate knowledge of the disk dimensions, the focal length of the lens, the axis of rotation, and the position of the detector would be required to determine a plane equation relative to a standard reference frame. Moreover, one might think that very complicated geometric computations would be involved. As we shall see in a later section, all of the necessary geometric unknowns can be expressed in a single matrix problem. Moreover, the requisite geometric measurements can be performed at once by a simple calibration procedure using only the known positions of seven or more lights. The calibration procedure directly determines not only the relative positions and orientations of the detectors in the room, but also the effects of the focal length of the objective lenses, and the positions and orientations of these lenses relative to the associated axes of disk rotation.

Some reference time must be established if the time at which light strikes the photomultiplier is to be converted into a slit position. An auxiliary photodetector with a fixed light source determines the time at which a slit reaches a reference position during each scan. This reference assembly also provides an input for measuring variation in rotational speed.

Two detectors share each rotating disk. The two detectors are placed roughly 90° apart around the periphery of the disk so that their scanning planes are approximately at right angles to one another. The two detectors are placed near the lower left and lower right parts of a disk. With the detectors mounted just below ceiling level (Figure 3), each detector can view a large volume of the room. A pair of detectors is mounted in each upper corner of the room which measures roughly twenty feet on a side. Wide angle lenses with fields of view of approximately 90° provide complete coverage of the volume of the room.

LIGHT SOURCES

The high switching rate required to turn on each light for only one scan virtually requires that light-emitting diodes be used. Since the duty cycle of each light is relatively low, the light-emitting diodes can be severely overdriven. Unidirectional conductivity permits an 8×8 array of light-emitting diodes to be driven with just 16 drivers (Figure 4). A lamp pack with the necessary drive circuitry and jacks for eight groups of eight lights each has been assembled. The lightweight lamp pack may be worn on a user's belt (Figure 5).

There is a tradeoff between scanning rate, resolution, and the amount of light which falls on a photocell. To get a high scanning rate, the disk turns at 3500 rpm. To get high resolution, we have made the slits quite narrow, 0.3 mm in a 35 mm image. As a result, very little light actually gets from a single lamp into the photocell. Adequate sensitivity in solid state photodetectors was not available, so photomultipliers are used. Efforts to maximize lamp brightness and photomultiplier sensitivity have payed off in acceptable performance. Of course, a design using electronic rather than

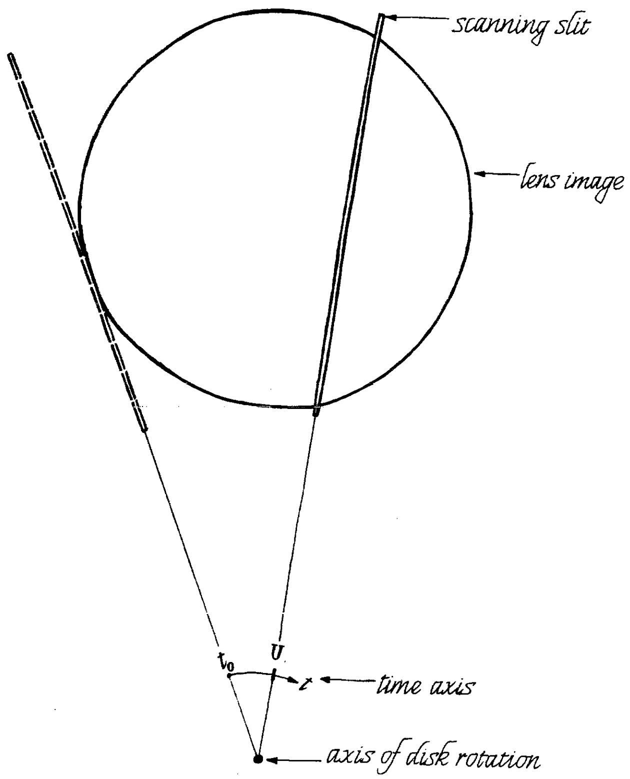


Figure 6—Perspective projection from the lens image into time space

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 \\ X_2 & Y_2 & Z_2 & 1 \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ X_n & Y_n & Z_n & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \\ \vdots & \\ t_{41} & 1 \end{bmatrix} = \begin{bmatrix} w_1 & & & & \\ & w_2 & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & \cdot \\ & & & & & w_6 \end{bmatrix} \begin{bmatrix} U_1 & 1 \\ U_2 & 1 \\ \cdot & \\ \cdot & \\ \cdot & \\ U_n & 1 \end{bmatrix}$$

Figure 7—Calculation of the transformation matrix T

or

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & -X_1U_1 & -Y_1U_1 & -Z_1U_1 \\ X_2 & Y_2 & Z_2 & 1 & -X_2U_2 & -Y_2U_2 & -Z_2U_2 \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ X_n & Y_n & Z_n & 1 & -X_nU_n & -Y_nU_n & -Z_nU_n \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{41} \\ t_{12} \\ t_{22} \\ t_{32} \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \\ \cdot \\ \cdot \\ \cdot \\ U_n \end{bmatrix}$$

Figure 7(b)

or

$$\begin{bmatrix} \\ X \\ \\ \end{bmatrix} \begin{bmatrix} T \end{bmatrix} = \begin{bmatrix} \\ U \\ \\ \end{bmatrix}$$

Figure 7(c)

mechanical scanning would have to face these same sensitivity problems.

COORDINATE CONVERSION

A simple matrix formulation of the computation of Cartesian coordinates from the time measurements is possible because a perspective projection is involved. Obviously the projection from the room space through the lens into the plane of the disk is a perspective projection. We can think of this image as being further projected into the space of time measurement by a projection centered at the axis of the spinning disk (Figure 6). This is again a perspective projection. In fact, the total projection from room coordinates into time space is a perspective projection. Using homogeneous coordinates it may be represented as

$$[X\ Y\ Z\ 1] \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} = [x\ y\ z\ w] \qquad (1)$$

and

$$U = \frac{x}{w}$$

where $[X\ Y\ Z\ 1]$ represents the room coordinates of a light, the t_{ij} are terms in a matrix related to the position, orientation, and dimensions of a detector, $[x\ y\ z\ w]$ are intermediate variables, and U is the time measurement made for that light.

Only a single measurement is made on the final perspective projection. Hence, only a single output variable U exists. The intermediate variables y and z play no part in the expression and these two columns of the matrix are irrelevant. As we learned in digitizing photographs[7] it is convenient to drop

$$\begin{bmatrix} & X^t & \end{bmatrix}\begin{bmatrix} \\ X \\ \\ \end{bmatrix}\begin{bmatrix} T \end{bmatrix} = \begin{bmatrix} & X^t & \end{bmatrix}\begin{bmatrix} \\ U \\ \\ \end{bmatrix}$$

or

$$\begin{bmatrix} X^t X \end{bmatrix}\begin{bmatrix} T \end{bmatrix} = \begin{bmatrix} X^t \\ U \end{bmatrix}$$

or

$$\begin{bmatrix} T \end{bmatrix} = \begin{bmatrix} X^t X \end{bmatrix}^{-1}\begin{bmatrix} X^t \\ U \end{bmatrix}$$

Figure 8—Least mean squared error fit

these two columns and rewrite the expression as:

$$[X \; Y \; Z \; 1]\begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \\ t_{31} & t_{32} \\ t_{41} & t_{42} \end{bmatrix} = w[U \; 1] \qquad (2)$$

and

$$U = \frac{x}{w}$$

which can be written

$$t_{11}X + t_{21}Y + t_{31}Z + t_{41} = U(t_{12}X + t_{22}Y + t_{32}Z + t_{42}). \qquad (2a)$$

The final expression can be thought of in three ways, depending upon what is known. If we know the position of a detector (the t_{ij}) and the position of a light ($X \; Y \; Z \; 1$), we can use this expression to compute the time U (a useless computation). If we know the position of the light ($X \; Y \; Z \; 1$) and the time measurement U we can compute the coefficients of one equation involving the unknown elements of the matrix T, and have the basis of a calibration procedure. Finally, if we know the position of a detector (the t_{ij}) and a time U, we can equally well compute the coefficients of a plane equation involving X, Y, and Z.

$$(t_{11} - t_{12}U)X + (t_{21} - t_{22}U)Y + (t_{31} - t_{32}U)Z + (t_{41} - t_{42}) = 0$$

or

$$aX + bY + cZ + d = 0 \qquad (2b)$$

This is the basic equation for coordinate conversion.

The calibration procedure deduces the elements of the matrix T from the times at which seven or more lights with known $X \; Y \; Z$ coordinates were sensed. The system of equations is shown in Figure 7. Since the scale factor of the t_{ij} is arbitrary, one of the t_{ij} may be specified and seven equations suffice. When more than seven reference lights are used, a least mean squared approximation is made to the resulting system of equations, as shown in Figure 8. This additional input avoids ill-conditioning in the system and reduces the effects of errors in the measurement of the positions of the reference lights. Since the values of the t_{ij} do not change, the calculation of the t_{ij} and the 7×7 matrix inversion implied in Figure 7 need be performed only once.

The coordinate conversion procedure to determine the three-dimensional position of a light source involves the simultaneous solution of the plane equations determined by each detector. As light sources move about the room each is visible to different detectors. As long as at least three detectors see a light, three plane equations can be determined and the position of the light can be deduced. If more than three detectors see a light, a least mean squared error fit can be computed.

$$\begin{bmatrix} \Sigma a_i^2 & \Sigma a_i b_i & \Sigma a_i c_i \\ \Sigma b_i a_i & \Sigma b_i^2 & \Sigma b_i c_i \\ \Sigma c_i a_i & \Sigma c_i b_i & \Sigma c_i^2 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \Sigma a_i d_i \\ \Sigma b_i d_i \\ \Sigma c_i d_i \end{bmatrix} \qquad (3)$$

where the a_i, b_i, c_i, and d_i are the plane coefficients for the ith detector. For example, $a_i = t_{11} - t_{12}U$ where the t's are elements of the previously computed matrix for the ith detector, and the time U is the time measured by the ith detector. Note that the plane coefficients are simply linear combinations of the t's and U, and thus are easily found from time measurements.

Because of the arrangement of detectors and the least mean squared error fit which accommodates redundant information, no difficulties are caused by parallel or nearly parallel planes. Most of the calculation time is involved in computing the summations of Equation 3. Note that we have not incurred the difficult calculations which one might expect to be associated with the complex motion through space of several scanning planes determined by several detectors whose positions, orientations, and dimensions are arbitrary. In practice, however, even the simple calculations which are required cannot be handled by a general-purpose computer sufficiently fast to keep up with the very many simultaneous measurements of lights. Equipment appropriate to handle these computations could be built easily. At present the problem is solved by generating information no faster than it can be handled.

OPERATING EXPERIENCE

In the short time that the Twinkle Box has been in use, several demonstrations of its capability have been made. No particular technical ability has been required to use the device. Real-time sensing and conversion of a single point source of light has been shown to be quite practical. However, no practical applications have been made. Positions of multiple lights have been determined in real time at a rate of 61 points per second. Data recording for 925 light positions per second with off-line computation of Cartesian coordinates has also been accomplished. The accuracy of the system has been determined by moving lights about the room at fixed distances from one another, and measuring separation. The standard deviation of the error from zero has been determined to be 7.3 mm, due primarily to time jitter on the photomultiplier pulses.

The design has three major deficiencies resulting from the mechanical scanners. First, in order to spin the 22-inch disk at 3500 rpm, a two-horsepower motor is required. A great deal of noise, vibration, and heat is generated. With four motors running, the room becomes unpleasant to work in. Second, the high starting load presented by disk windage in a housing which does not fit tightly about the disk forced us to use induction motors. Because the disks do not rotate phase-locked, each light must be activated for the time period of two scans (one millisecond). This guarantees that the light is on at the beginning, during, and at the end of at least one complete scan by each detector. Were the disks to rotate phase-locked, we could double the data collection rate. Finally, in spite of considerable care in the production and assembly of the disks, pulse times determined by different slits for a stationary light are distributed in time. Because the pattern of distribution repeats for each revolution of the disk, we attribute it to nonconcentricity of the disk and motorshaft, and to errors in slit placement. A correction table has been built and is used at run-time after several sequential reference pulses are received. A design allowing for continuous input by each reference sensor, to indicate which slits are scanning the images of the room, would obviate this deficiency.

There is considerable optical distortion in the wide-angle lenses chosen for the detectors.* This distortion causes inaccuracies in the measurement of positions. The magnitude of these inaccuracies is a function of the position of a light source relative to the calibration lights and relative to the optical axes of the objective lenses. Measurements near the calibration lights and near the intersection of several optical axes are quite accurate; accuracy deteriorates with increased separation. Efforts have been made to correct lens distortion. However, only a single dimension is measured by each detector. Unless corrections are made using data from more than one detector, these efforts can at most decrease the magnitude of errors by a factor of three.

An additional source of error is attributed to the fact that different detectors may see a light at different times. In this case, nonsimultaneous data are gathered. Similarly, different lights are measured at different times. Obviously, the actual time at which a detector sees a light is known, as is the time at which various lights are measured, for this is the basic measurement. Some correction might presumably be made.

* Vivitar Auto-Preset, f.l. 20 mm, F3.8 lenses are used. Linear distances are diminished up to 5.89 percent by regular distortion at the edge of a lens and up to ±1.2 percent by irregular, randomly located distortion.

POSSIBLE FUTURE DEVELOPMENTS

Henry Fuchs* has discovered that reflected light from a laser beam provides an adequate input to the Twinkle Box detectors. It is possible, therefore, to think of a device which would deflect a laser beam in two dimensions. The Twinkle Box could then sense the reflected light in one dimension. Such a device could easily measure the three-dimensional profiles of objects such as people's faces, and thus provide a new form of input. Two-dimensional deflection of a laser beam using mirror galvanometers appears to be quite practical. Since the Twinkle Box detectors are placed about the room, the object(s) to be scanned could be positioned at random.

It is possible to think in terms of reversing the Twinkle Box detectors and light sources. In such a reversed system, several one-dimensional scanning projectors would provide illumination in sequence to small photocells whose positions would be measured. Each projector would sweep the working volume with a plane of light moving in one dimension, and each photocell would report the times at which it sensed light. The mathematical computations required for coordinate conversion would be identical to those for the Twinkle Box. There are several possible advantages to such a reversed system. First, as much light as desired could be transmitted. Second, because parallel sensing of light pulses at each photocell could be provided, a much lower scanning rate could be used to measure as many positions as there are photocells. Using simpler scanners, scanning rates of 100 per second rather than 2000 would suffice. Finally, because no room image would be formed in a detector, the need for wide angle lenses with their associated distortion could be avoided. Light columnization could be accomplished by baffles and lenses. The disadvantage of the reversed system is the increased electronic complexity of providing 50 to 100 photocells each capable of reporting the time of a light pulse. Modern solid state photocells, amplifiers, and integrated circuitry appear, however, to make such a design worth serious consideration.

* University of Utah.

BIBLIOGRAPHY

1. Brenner, F. H. and M. T. Zayac, *A Multi-Color Plasma Panel Display*, Owens-Illinois, 1971.
2. Burton, R. P., *Real-Time Measurement of Multiple Three-Dimensional Positions*, University of Utah Computer Science Technical Report UTEC-CSc-72-122, June 1973. Also Technical Manual TMAN-73-01. (Supported by ARPA Contract F30602-70-C-0030.
3. Capowski, J. J., *Remote Manipulators as a Computer Input Device*, Department of Computer & Information Science, University of North Carolina (Chapel Hill), 1971. (Supported by AEC contract #AT-40-10-3817.)
4. Curry, J. E., "A Tablet Input Facility for an Interactive Graphics System," *Proceedings International Joint Conference on Artificial Intelligence*, Walker, D. E., L. M. Norton, eds. May, 1969.
5. Davis, M. R. and T. O. Ellis, "The Rand Tablet: A Man-Machine Graphical Communication Device," *FJCC 1964*, Spartan Books, Baltimore, Md.

6. Noll, A. M., "Man-Machine Tactile Communication," *SID Journal*, July/August, 1972. Also Ph.D. dissertation, Department of Electrical Engineering, Polytechnic Institute of Brooklyn, 1971.
7. Parke, F. I., *Computer Generated Animation of Faces*, University of Utah Computer Science Technical Report UTEC-CSc-72-120, June 1972.
8. Roberts, L. G., *The Lincoln Wand*, M.I.T. Lincoln Laboratory Report, Lexington, Mass., June, 1966.
9. Science Accessories Corporation, *Graf/Pen Sonic Digitizer*, Science Accessories Corporation, Southport, Conn., 1970.
10. Stoutemyer, D. R., *Systems Study and Design of a Blind Mobility Aid Simulator*, Master's thesis, Department of Mechanical Engineering, M.I.T., 1965.
11. Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communication System," *Proceedings of the Spring Joint Computer Conference*, Detroit, Mich. May 1963, and *M.I.T. Lincoln, Laboratory Technical Report No. 296*, January, 1963.
12. Sutherland, I. E., "A Head-Mounted Three-Dimensional Display," *AFIPS Conference Proceedings*, Vol. 33, pp. 757-764, December, 1968.
13. Teixeira, J. F. and R. P. Sallen, "The Sylvania Tablet: A New Approach to Graphic Data Input," *SJCC 1968*, Thompson Books, Washington, D.C.

APLG–An APL based system for interactive computer graphics

by W. K. GILOI and J. ENCARNACAO

Universitat des Saarlandes
Germany

INTRODUCTION

Computer Graphics (CG) is in its essence interactive. The customary "batch processing" languages such as, for example, FORTRAN, ALGOL, PL/I, . . ., are not primarily designed for interactive use. One of the main reasons for that is their relatively early binding time. Another handicap, particularly for CG, is a certain inadequacy of their control structure for the kind of dialog characterizing CG. Thus, the usual programming packages for CG based on one of the above mentioned high-level languages have to be augmented by devices which allow the system to lend control to the user, together with a simple command language which enables the user to execute temporarily control over the system. The commands are given to a command language processor which works interpretatively, initiating, for instance, respective subroutine calls.

The basic transformations of objects (translation, rotation, scaling, perspective, etc.) are vector or matrix operations. This holds especially in the case of display systems in which straight-line segments are used as graphical primitives (so-called vector displays), and so far most computer graphics display terminals are intrinsically line drawing. Thus, a host language for CG should have vectors and matrices as data types and arithmetical operations on arbitrary sets of arrays or operations to rearrange arrays as primitive operations.

Thus, we feel that a widely used interactive, high-level programming language which provides the following features

—Avoidance of formal operand declaration combined with an interpretative execution, leading to a very delayed binding;
—Control structure for interactive use;
—Inclusion of arrays as primitive data types

will be a better "host" for a language extension for computer graphics than one of the above mentioned languages. Such a language is Iverson's APL.[1] Consequently, the extension of APL for CG has been repeatedly suggested, and APL is in fact in use at several places for the purpose of computer graphics. In this paper, we shall outline the general philosophy of such a language extension, using the possibility of APL to write proprietary functions for special purposes. Whereas this alone would not justify to use the term "language extension," the justification will stem from the fact that the APL interpreter will have to be truely extended by some additional primitives to be explained in this paper.

Such an extension of ALP, called APLG,[2] will be proposed in this paper. Furthermore, we shall show a way for a very efficient implementation of such a system, meeting the given constraints of a time sharing environment. Moreover, we hope that this paper may also contribute in general to the ongoing discussion about the question what kind of intelligence to put into an intelligent terminal.

THE ORGANIZATION OF APLG

If one starts out to extend APL into a graphic language, it is a very tempting idea to add to the standard repertoire of monadic, dyadic and mixed functions an additional set of complex operators for construction and manipulation of display items. Such an approach to the language extension —however desirable—has a serious disadvantage, namely that such a solution would require a change of the syntax of APL and, hence, the implementation of a different APL interpreter. Our goal, however, is to permit the use of the existing APL implementations and extend it for graphics strictly by adding a few additional primitives. In pursuing that goal, APLG will contain the existing ALP as a subset.

Our goal can be achieved if the language extension is restricted mainly to the definition of a set of standard functions to be added to the existing APL interpreter. These functions take care of all operations required for interactive graphics: picture generation and manipulation as well as display file management. An extension of APL to APLG requires, thus, only the augmentation of the existing translator by the respective procedures—and, of course, the extension of the set of recognizable primitive functions. The standard functions which cause the drawing of items or control the light pen, etc. will generate code in an intermediate language to be interpreted by the respective display processor in the graphics terminal and, hence, the portability of such a system is ensured. Other standard procedures for the transformation of pictures may strictly be kept within the proper APL body.

The extension of ALP to APLG has to provide the ability to create and manipulate "pictures" and to work interactively at a display console. In order to display a picture on a CRT screen, code which can be interpreted and executed by the display processor in the terminal must first be produced. In extension of the standard APL environment, we add a second input/output medium, namely the display console, and hence, we have now to distinguish between results of a function application which are obtained in the normal calculation mode of APL, i.e, which are typed out, and results which have to be displayed on the CRT screen. This can be accomplished by putting the latter into a particular memory area which is usually called the "display file."

Following KATZAN's philosophy,[3] we build up the display file by defining three special APL arrays with the reserved names *DC* (display code file), *NL* (name list), and *CT* (correlation table). Here and in the following, reserved names will be indicated by *italics* in order to make it very unlikely that a user might use the same name for a user's function (an allowable technique in APL). As the names say, the *DC* array will accommodate the display code, the *NL* array will contain the names of the pictures and those of the picture items defined by the display code, and the *CT* array will establish the relationship between names stored as character strings in *NL* and the graphics program stored in *DC*. *CT* is a two column matrix, from which can be deducted what name is assigned to a picture or a graphical item (part of a picture). The rows of *NL* and *CT* are related one-to-one to each other, e.g., such that an ith row of *NL* corresponds to the ith row of *CT*. The following example will illustrate the construction of the display file.

As an example, we assume that the program has to generate the picture shown in Figure 1A. This picture consists of the two points P_1 and P_2, a straight line connecting them, and the point IS on this line which is the intersection point with a (not visible) line $\overline{P_3P_4}$. The program calls a number of standard functions of APLG which will be explained later. The name of the picture shall be PIC.

Example 1:

```
      PROG [⎕]→∇
    ∇ PROG; P
[1]   FILE
[2]   PICTURE 'PIC'
[3]   P ← 2 2 ρ A
[4]   'P' POINT P
[5]   'L' POLYGON P
[6]   'IS' POINT A ISECT B
[7]   END 'PIC'
    ∇
```

The program is actually executed and the result displayed on the CRT screen by specifying the operands A and B, followed by calling a function *S*HOW. The Parameter S of *S*HOW is a "status vector" which defines the mode of display.

Program call:

```
A←100 100 500 600
B←200 450 400 80
PROG
S SHOW 'PIC'
```

During the execution of PROG, the standard function *F*ILE (see label [1] in the program) generates the *NL*, *CT*, and *DC* arrays, i.e., the display file for this program as shown in Figure 1B. *F*ILE is called only once before storing the graphics code in the display file. The picture name PIC is brought into *NL* by the *P*ICTURE function. Since the picture consists in this case of three items: the set of points P = {P_1, P_2}, the line L, and the intersection point IS, the name list has three more entries following the picture name, that is to say these item names.

The first row of the correlation table *CT* contains two pointers: the first pointing to the beginning of the display code file *DC* and the second pointing to the end. The first pointer is entered by the *P*ICTURE function and the second by the *E*ND function. In such a way, the display code file is built up dynamically and can have any arbitrary length. A negative integer in the first cell of a row of *CT* indicates that this row corresponds with a graphic item. The value −1 means ⟨point⟩, for example, and −2 means ⟨line⟩. This simplifies the identification of the types of items. The second value indicates the number of items which are represented by one item name.

Generally speaking, the APLG display file is structured in the form of a 3-level tree. The lowest level is given by the graphic primitives, points and lines, the next higher level is given by collection of primitives of the same type, called items, and the root of the tree is given by a collection of such items, called picture.

The user does not know the exact extent of his display file (and does not have to). The only indication of the existence of a display file in his workspace is given by the fact that the usable part of his workspace shrinks with a growing display file, and a workspace overflow message may occur sooner. If it is considered to be necessary, the *F*ILEs function which builds up the workspace may also perform some garbage collection.

The statements of the program body are all standard function calls, except the operation which reshapes A into

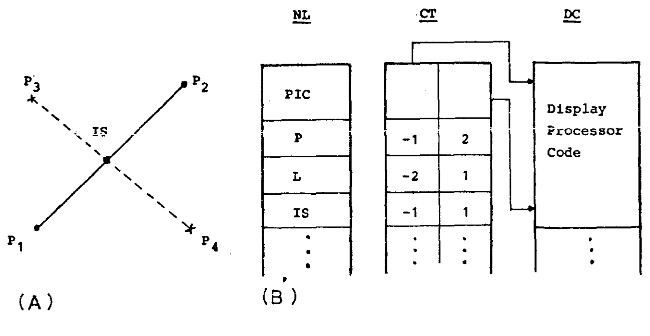


Figure 1—(A) Illustration of example 1; (B) Resulting display file

a matrix. *P*OINT generates the graphics code for a specified number of points and *P*OLYGON the one for a polygon. *I*SECT calculates points of intersection of specified lines. Nesting of several pictures is possible.

Note that the difference between a usual function definition in APL and the definition of a function for the generation of a picture is the following:

(1) the picture definition part has to begin with the standard function *P*ICTURE and to end with the call of the standard function *E*ND with the picture name as parameter, and
(2) picture and item names are defined as character strings, which are put between quotation marks.

The *E*ND name statement provides the possibility of nested picture definitions. The special characterization of picture and item names is necessary in order to have them recognized as such and not as variables.

The function execution and the generation (and subsequent display) of the display code is actually started by the *S*HOW function, but of course not before the function has been supplied with the required data. The 'status' vector S has the following dimension.

S→h i j k

The components h, i, j, k may have, for example, the following meaning:

h : blink status	: h = −1 picture blinks	
	h = 0 no blinking	
i : beam intensity	: i = −1 picture invisible	i = 0.5 dim
	i = 0 normal intensity	i = 1 bright
j : style	: j = 0 solid lines	j = 1dashed lines
	: j = 2 dotted lines	j = 3 = dashed-dotted lines
k : light pen control	: k = 0 light pen disabled	
	k = 1 light pen enabled	

Part or all of these parameters can be omitted, beginning from the right-hand side. Any omitted parameter is automatically interpreted as zero. Thus, if S is 0, it means: no blinking, normal intensity, solid lines, light pen disabled, while S←−1 would mean the same status except that the picture is now blinking. Note that the light pen pick status (light pen pick enable/disable) can in such a way only be defined for a whole picture but not for individual items of a picture. Therefore, a second way of specifying the light pen pick status is provided by the *E*NABLE function (see next section) which can be inserted into a picture definition function. The status specified by *E*NABLE overrules the one specified in S. Note that the purpose of the *S*HOW function in connection with the parameter S is to provide an easy way to change arbitrarily the status of a picture representation, that is *S*HOW can be called many times in the course of a terminal session with a varying status.

A COLLECTION OF STANDARD FUNCTIONS OF APLG

In the following we will list a representative selection of APLG standard functions (Table I). In our opinion, APLG should always be considered as a concept open for extensions and with enough degrees of freedom to satisfy the particular requirements imposed by the individual display systems. That means that—besides a core of standard functions evolving over time—there should be always the possibility of adding special library functions proprietary for a given system.

So far we have been considering the display solely as an output device. We shall now discuss some functions which allow the input of graphic information and, thus, man-machine-interaction.

The following example shows how to place ten dots on the CRT screen (and input their coordinates) by use of the light pen. The *R*ASTER function generates a grid which, in conjunction with the light pen, causes an interrupt whenever a grid line crosses the light pen aperture. After the coordinates of the picked raster points have been determined by *C*OORD, a dot is displayed at the proper spot on the screen. All these points are stored under a common name, P.

Example 2:

```
      ∇LIGHTPEN[⎕]
    ∇ LIGHTPEN
 [1]  PAUSE
 [2]  →SOURCE×1
    ∇
      ∇POINTIN[⎕]∇
    ∇ POINTIN
 [1]  I←O
 [2]  ATTENTION 1
 [3]  PICTURE 'PIC'
 [4]  0 1 SHOW 'PIC'
 [5]  K:RASTER
 [6]  LIGHTPEN
 [7]  PO←COORD
 [8]  I←I+1
 [9]  →L×₁I≠1
[10]  'P' POINT PO
[11]  →K
[12]  L:''POINT PO
[13]  →K×₂I≠10
[14]  END 'PIC'
    ∇
```

The coordinates of a primitive, the name of the picture and item to which it belongs, and its index within the item shall be determined after a light pen pick of this primitive. This can be done using the *P*ICNAME, *I*TEMNAME,

TABLE I—APLG—Standard Functions

Function name	Operation
*F*ILE	Organization of a display file
*P*ICTURE 'pic'	Start of picture definition
*E*ND 'pic'	Closing of picture definition
S *S*HOW 'name'	Starts display of 'name' with status S
a *D*ELETE 'name'	Removal of a picture or an item or all items with a common name from the display file $a=0\wedge\text{'name'}\neq''$ deletion of 'name' $a=N(N>0)\wedge\text{'name'}=''$ deletion of item with index N in *DC*
'item' *I*NDEX 'pic'	Returns the index of an item in 'pic' (absolute index in *DC*)
*I*TEMNAME	Retrieval of the item name of a picked item. Explicit result: item name
*P*ICNAME	Retrieval of the picture name after a light pen pick. Explicit result: picture name
*C*OORD	Returns the coordinate values of a primitive after a light pen pick (if the primitive is a line, the coordinate of the terminal point are returned). Explicit result: x y
*R*ASTER	Generation of a raster for light pen indication of a position
b *E*NABLE 'name'	Enable/disable light pen pick of an item or picture (overrules the specification given in the status vector S); b=0 disable; b=1 enable b can also be a boolean vector representing a mask for the item names of a picture
*A*TTENTION a	Indicates to the system which attention sources may occur: a=0 no attentions a=1 light pen a=2 keyboard · · a=32 all attention sources, etc.
*S*OURCE	Returns an integer value indicating the attention source (e.g.: 1 for light pen attention, 2 for keyboard, etc.)
*P*AUSE	Computer waits for an attention. Picture regeneration continues.
'name' *P*OINT ⟨point⟩	Generation of a set of points with a single name
'name' *P*OLYGON ⟨point⟩	Generation of a polygon
'name' *T*EXT ⟨text⟩	Display of text (array of characters)
*P*OSITION ⟨point⟩	Initialization of the beam position
*S*UPPRESS	Stop picture regeneration without clearing the display file (renewed display of a picture can only be achieved by use of the *S*HOW function)
*R*ETURN	Return to teletype control
point TRANSFORM M	Transformation of a set of points $M=3\times3$ matrix (two-dimensional case) 4×4 matrix (three-dimensional case) Explicit result: transformed points
'pic' *P*ICTRANS M	Transformation of a picture

and *I*NDEX function. *E*NABLE is called to enable the light pen pick of any item of the picture independent of the status specified in the respective *S*HOW instruction assuming that the picture is already being displayed.

THE ENVIRONMENT FOR AN IMPLEMENTATION

The environment for the implementation of an interactive software system—such as APLG in the case we are concerned with—is on one hand determined by the required properties of this system—first of all by the ways in which files have to be organized and updated—and on the other hand provided by the technical properties of the system.

It is obvious that, in a computer-terminal-dialog, both sides must have access to the display file. The application program which generates the display file in the first place has to update it whenever a change in the pictorial representation on the display screen shall occur. The terminal, on the other hand, which receives the display file as the program for its display processor, may cause an updating of the display file, too, whenever its user wants some changes in the picture on the screen. We will call any action which changes the pictorial representation on the CRT screen "picture editing."

We conceive an "intelligent" terminal which serves the user's interactions locally and communicates, on the other hand, with the time-sharing system. If such a system is organized the way it will be proposed in the following, the internal processor of such an intelligent terminal has to be primarily a list processor. All necessary functions of such processor can be easily realized as firmware routines executed by one of the now available, inexpensive MOS/LSI microprocessors. With such a system, it will become feasible to provide interactive computer graphics services to a larger community of users in a very economical way.

In regular time-sharing systems, the communication links between central computer and terminals have to use voice grade telephone lines for reasons of economy and, hence, their transmission rate is rather low. This is no serious handicap if the terminal is a teletype or a storage

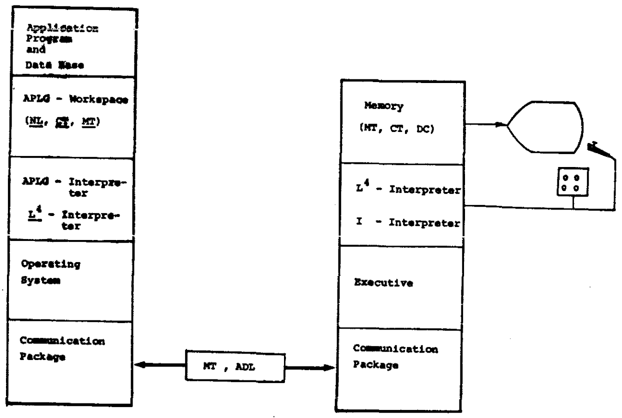


Figure 2—Component of the APLG time-sharing system

tube display, but it would not be tolerable if the terminal is an interactive graphics display with a dedicated computer, and if large amounts of information have to be swapped between the two computers in the system. On the other hand, in the time-sharing system for computer graphics services which we have in mind, high-speed communication links have to be ruled out as being to costly, and that puts a very serious constraint on the design of the system. Hence, we have to devise organizational means by which the amount of information that has to be exchanged between central computer and terminal will be minimized.

THE INTRA-SYSTEM COMMUNICATION

At the beginning of a dialog between the user at the terminal and the application program in the central computer, a display file assembled in the user's workspace by his APLG program has to be transmitted to the terminal leading there to the generation of a pictorial representation on the CRT screen. In the course of a dialog, messages will have to be exchanged between central computer and terminal whenever the user or the application program ask for certain actions or respond to certain requests made by the other side. The medium for such a dialog is always the pictorial representation on the screen. Thus, man-machine interaction means primarily "picture editing," and hence, —as a picture always has its logical representation in the display file—display file manipulation.

A closer look reveals that only part of the information in the display file is relevant for both sides in the dialog. On one side, names are only meaningful to the application program and, thus, the name list does not have to be transmitted to the terminal. On the other side, the display processor code is only relevant for the terminal. Therefore, we may decompose the display file into two intersecting parts, one part stored in the main computer and containing the name list and the correlation table, and the other part stored in the terminal and containing a copy of the correlation table and the code list. The intersection of both is the correlation table which establishes the link between the two separated parts.

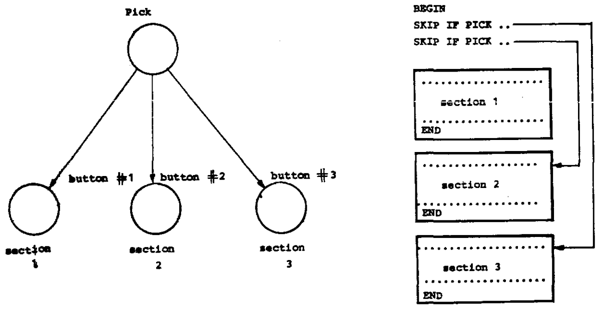


Figure 3—"Linearization" of a branching in an interaction handling program

Our philosophy is to generate the total display file just once at the beginning of a dialog and update it from then on successively. That way, only the (non-redundant) information concerning the picture editing has to be exchanged. To this end, we need a vehicle which will carry this information back and forth, as editing may be done from either side. Such a vehicle will be provided by two additional lists, one called *'message table'* MT and the other *'associated data list'* ADL. These messages will be formulated in an intermediate language especially designed for that purpose and named L^4.

Thus we obtain a system as illustrated in Figure 3. On one side we have the central computer with its operating system and an *APLG interpreter** (which includes, of course, an APL interpreter). Every user APLG has his APL workspace which contains besides his programs his proprietary functions and his data. The workspace contains, furthermore, a part of the actual display file, namely the name list *NL* and the correlation table *CT*. The *APLG interpreter* encompasses among the other standard APLG functions[2] a particular function called *L^4 interpreter*.

The terminal, on the other hand, contains also some parts of the actual display file, namely a copy CT of *CT* and the display processor code list DC. Additionally, there is a second L^4 interpreter—which differs essentially from the one to be found in the APLG translator—and a module called *"interaction handler"* (IH) and to be explained later. MT is a matrix that has three columns and an arbitrary number of rows. The entry in each row constitutes a statement in a special intra-system communication language called L^4. ADL is a linear order of data blocks, to which pointers in the L^4 statements give access.

THE INTERMEDIATE LANGUAGE L^4

Messages sent from the central computer to the terminal have to serve the purpose of generating display processor code, supplying data, and edit the correlation table CT. Conversely, messages sent from the terminal to the central processor may supply data and edit *CT*, too, but also the name list *NL*, and they have to inform about the source and the attached parameters of attentions. All these purposes are served by the command language L^4 defined in the following. *Nota bene*, L^4 is not a programming language, and the user of the system is not concerned with it, but nevertheless, it may be useful if we use a programming language type mnemonic notation. If the level of a language is measured by the cardinality of the set of functions and data types occurring in that language, the language we need has even a lower level than the customary assembly language and, hence, we call it a low-low-level language and name it, therefore, L^4.

* Names of software modules in the central computer will be distinguished from names of software modules in the terminal by underlining.

TABLE II—List of L^4 Statements

Group 1:	Display processor code generating commands
	SET CONTROL, O, reference
	SET MODE, O, reference
	SET STATUS, O, reference
	INITIALIZE, O, reference
	DRAW SHORT POINT TO, O, reference
	DRAW SHORT LINE TO, O, reference
	DRAW SHORT BLANK TO, O, reference
	DRAW LONG POINT TO, O, reference
	DRAW LONG LINE TO, O, reference
	DRAW LONG BLANK TO, O, reference
	DRAW LONG POINT AT, O, reference
	DRAW LONG LINE AT, O, reference
	WRITE TEXT, O, reference
Group 2:	Display file manipulating commands
	DELETE, i, reference
	INSERT PICTURE, i, reference
	INSERT ITEM, i, reference
	NEWDATA, i, reference
	NEWNAME, i, reference
	NEWSTATUS, i, reference
	DATA, n, reference
Group 3:	Interaction handling commands
	WAIT FOR PICK, O, O
	WAIT FOR HIT, O, O
	WAIT FOR POINTING, O, O
	SKIP IF PICK, k, reference
	SKIP IF HIT, k, reference
	SKIP IF POINTING, k, reference
	BEGIN CREATE QUEUE, n, reference
	QUEUE IN, O, O
	INSERT QUEUE, i, n
	ENABLE, s, O
	DISABLE, s, O
	END, O, O
Group 4:	Attention source indicating messages
	PICK RETURN, s, reference
	HIT RETURN, s, reference
	POINTING RETURN, s, reference

L^4 statements are of the following form

COMMAND, INDEX, REFERENCE

Semantics:

COMMAND
specifies the operation to be performed.

INDEX
is an integer which gives the index number of the CT row that is associated with the entity (picture or item) subject to the operation.

REFERENCE
is a pointer to the entry in a data list ADL where the data block associated with the above command begins. The delimiter for this data block is the first non-zero reference found in the following statements.

The index origin of all tables is One. If a command does not apply to an object or does not require a data reference, the respective parameter is set to Zero.

It should be emphasized that the CT row index (or *CT* row index, respectively) used for object specification is sufficient to retrieve the associated name from the name list as well as to provide access to the associated display processor code. COMMANDS may be grouped into three categories as listed in Table II.

Semantics:

(1) 'TO' specifies coordinates as being increments, whereas 'AT' refers to absolute coordinates.
(2) In the DATA command, n identifies a data block to the application program.
(3) BEGIN . . . END are used to delimit a linear section of the interaction handling program. WAIT IF causes the terminal processor to wait for an interrupt and report the respective action of the user. PICK denotes a light pen pick, HIT denotes a pressed key of the function keyboard, POINTING denotes the indication of coordinates by light pen, cursor, or tablet stylus. BEGIN CREATE QUEUE tells the terminal to reserve a number of memory cells, specified in the data part, under the name n and to create at the end of this section a report using the DATA statement of group 2. QUEUE IN is executed during a WAIT and puts all data supplied by the user (regardless of their source) into the queue. INSERT QUEUE tells the terminal to put the content of that queue into the display file part which belongs to entity i (thus, INSERT QUEUE has the same effect as a NEWDATA statement). ENABLE and DISABLE cause the terminal to enable or disable the specified attention sources (light pen, keyboard, tablet, etc.). SKIP IF combines the WAIT with a conditioned SKIP command. Whereas WAIT causes a report of the user's action, SKIP prescribes this action as the skip condition.
(4) PICK returns an object identifier; HIT returns a key identifier; POINTING returns a pair of coordinate values. 's' gives the attention source.

The reader will notice that the statements of group 1 act as an intermediate language for the display processor code (DPC), that is, APLG functions[2] such as, for example, *P*OINT or *L*INE, etc., which are used to generate pictorial objects, will not generate DPC directly but L^4 code. The translation of L^4 code into DPC will be performed by an interpreter in the terminal. This has the following advantages

(i) All L^4 statements have the same format independent of the group they belong to and, thus, we obtain a very uniform structure of L^4. Moreover, the information types of L^4 can be better matched than the information types of a typical DPC to the ones used in APLG.

(ii) The translation of certain L^4 statements may result in more than one DPC statement. Thus, L^4 provides a more concise way of coding display processor operations than the DPC itself.

(iii) The introduction of L^4 provides not only a way to convey information in a concise, non-redundant form, but it furthers essentially the portability of APLG (or any similar system).

The last fact should be pointed out more in detail. L^4 is constructed so that we have commands for all functions which a display processor may conceivably have. A display processor, on the other hand, may range from a simple plotter to a high-performance vector display. Thus, commands given to such a sophisticated vector display may be expressed by L^4 statements as well as commands given to a simple plotter (in the latter case, of course, only a small subset of L^4 will be actually used). As the interpreter which translates L^4 code into the DPC of a particular display system is part of this system, we have no problems to use the same software (such as, for example, APLG) for a large variety of different display media. We may consider such a terminal as being an abstract machine whose machine language is L^4.

THE "INTELLIGENT" TERMINAL

The intelligent terminal has a preprocessor and a display processor. The preprocessor has to perform mainly the following tasks.

(1) Interpretation of L^4 statements and translation into display processor code (DPC).
(2) Origination and updating of the part of the display file (CT and CL) accommodated in the terminal,
(3) Identification of picked light buttons or other objects, recognition of attention sources, and formulation of respective messages to the application program.
(4) Acceptance of input from the alphanumeric keyboard.
(5) Execution of given subprograms which enforce a certain sequence of actions by the user.

(i) L^4 interpretation and display file management

Due to the rigid structure and the very low level of the L^4 language, interpretation is very simple. The command code gives directly the entry to a table in which the start addresses of the command execution procedures (CEP) are stored. The CEPs may call for their part certain commonly used subprocedures. It has already been mentioned that a CT index provides access to the related piece of display processor code. The required operations are to count (to increment and decrement) and to add. Additionally, logical operations are necessary for bit masking and testing. All these functions can be easily performed by a minicomputer or even a MOS microprocessor.

(ii) Object identification

The object identification routines depend very much on the type of display terminal and on the fact whether such a terminal has a light pen or whether identification can only be done by positioning a cursor and test the coordinates of the cursor position. The required capabilities are again to count and add, but also to compare coordinate values.

The most crucial part of the envisioned computer-terminal-symbiosis is the handling of user's interactions. We emphasized already that for practical reasons we have to find a way to avoid the handling of single actions of the user by the central processor. Hence, interactions should be autonomously handled by the terminal. However, requests for a certain sequence of user's actions are made by the application program, and the sequence is determined by the logic of that program. We may conceive an attention handling program segment as a more or less complicated decision tree. If the whole tree is too complex to be handled in the terminal, we have to break it into subtrees which can be handled. The simplest case is to chop the entire tree into pieces such that each piece is a linear section connecting two branch nodes. In this case, testing and branching is performed by the application program, whereas all other steps are executed by the terminal processor.

As the reader may have already noticed, there is a device in the L^4 language which allows to transform such sections of an attention handling program (which is in APLG represented by a respective function) into L^4 code. This device is given by the BEGIN ... END delimiters. BEGIN signals to the terminal processor that the following L^4 statements form an entity which is a linear part of an attention handling routine enforcing a certain sequence of actions by the user. During the interpretation of such a 'program', the terminal processor will enter all the information gathered about the user's actions into a list of L^4 statements called 'message table' (MT). Eventually, on recognition of the END delimiter, this message table will be transferred to the central computer into a reserved field, called MT, and subsequently, MT in the terminal will be cleared.

This scheme of handling interactions is certainly very simple. It is unsatisfactory if the terminal processor is powerful enough to handle more than just the part between two branch nodes of the decision tree which represents an attention handling program. Therefore, we provide in L^4 the possibility to let the terminal execute larger subtrees.

Principally, the execution of a subtree requires a branching. As the simple structure of the L^4 language does not include labels, a GOTO is not feasible and hence, the only way to branch is to use a conditioned SKIP command together with a parameter k which gives the length of the skip. In L^4, the conditioned SKIP command is combined with a WAIT FOR EVENT process, where the event is an action of the user, that is, if the result of the user's action matches the condition set in the SKIP IF EVENT command, then the following k commands are skipped, else the next statement is executed. We would like to emphasize that this is the highest form of interaction handling in the terminal we dare recommend. Any more complex structures such as, for instance, recursive use of certain commands, nesting of attention handling program segments, etc. should be left to the program in the central computer to deal with.

As in any truly interactive system, control is switched back and forth between central computer and terminal. The central computer hands over to the terminal by sending a BEGIN statement, and the terminal hands control back—together with a report of the interactions that have taken place in the meantime—when the END statement occurs.

CONCLUSION

It is interesting to see how well our proposed system—which was in the first place designed for maximal efficiency under serious technical constraints—complies with a modern construction rule for software systems, namely to modularize such a system into several strata of languages and their interpreters. If L is a language and I is an interpreter for that language, the ordered pair (L,I) is sometimes called a computing system C. In our case, we have a stratified hierarchy of such computing systems such that, on the way from the application program in the computer to the user at the terminal, each interpreter generates productions in the language defined on the next lower stratum. Listing these computing systems in detail, we have:

Location	*Language*	*Interpreter*
Computer	*APLG*	*APLG-INTERPRETER*
Terminal	L^4	L^4-INTERPRETER
Terminal	DPC	DISPLAY PROCESSOR
Terminal	PICTURES	HUMAN INTERPRETER
Terminal	INTERACTIONS	INTERACTION INTERPRETER

There is also a way back from the INTERACTION INTERPRETER to APLG via L^4 as carrier of messages and the *L^4-INTERPRETER* in the central computer. However, this way back to APLG is not as straightforward as the way down from APLG, since the user's interactions are not context free but only interpretable if certain information from the APLG program and from the DPC program are available.

An implementation of the APLG system outlined in this paper is under way at the University of Minnesota. The environment for this is provided by a CDC 6400 time-sharing system running under the KRONOS operating system, for which an APL implementation was provided by W. Franta and G. R. Mansfield.[4] The terminal is an IDIIOM I display system of INFORMATION DISPLAYS INC. with a VARIAN 620 as 'controllable memory.' The communication link has only 300 baud (and this will really put our message table mechanism to test).

In order to ensure the feasibility of our concept, we wrote and tested most of the key functions of APLG. The intra-system communication on the base of L^4 was tested by elaborate simulations of a 'virtual' computer-terminal-system. The simulation proved a faultless operation of the respective PRADIS[5] module for interactive construction of 3-dimensional objects. The author's are greatly indebted to Mr. J. Hunger of the Heinrich-Hertz-Institute in Berlin/Germany who performed the simulation.

REFERENCES

1. Iverson, K. E., *A Programming Language*, J. Wiley & Sons, New York, 1962.
2. Giloi, W. K., J. Encarnacao and W. Kestner, "APLG—APL Extended for Graphics," *ON LINE 72*, Conference proceedings, Vol. 2, pp. 579-599, BRUNEL University, Uxbridge/England.
3. Katzan, H. Jr., "Representation and Manipulation of Data Structures in APL," *SIGPLAN Notices*, 6, 2, Feb. 1971, pp. 366-397.
4. Mansfield, G. R., *APL/KRONOS—An APL Interpreter for the CDC 6000 Series Computer*, M.Sc. thesis, University of Minnesota, CICS Dept., Minneapolis, Minn., 1972.
5. Encarnacao, J. and W. K. Giloi, "PRADIS—An Advanced Programming System for 3-D-Display," *Proc. AFIPS SJCC*, 72, pp. 985-998.

Project FIND—An integrated information and modeling system for management

by JOHN S. McGEACHIE and DONALD L. KREIDER

Dartmouth College
Hanover, New Hampshire

INTRODUCTION

Project FIND (Forecasting Institutional Needs for Dartmouth) has been established at Dartmouth College to make institutional data readily accessible to administrative officers and faculty members through the facilities of the Dartmouth Time-Sharing System (DTSS). A concomitant goal is to develop models of the operation of the institution to facilitate long-range planning by providing quantitative estimates of the effects of policy changes.

Thus Project FIND has a threefold purpose:

(i) to provide a tool, requiring minimal computer expertise, through which administrators and faculty members can obtain current information on the college's finances, students, staff, and physical facilities,
(ii) to provide a universal format and language through which members of the college community can manage their own private data, and
(iii) to provide a capability for making projections of expense revenue trends, tenure ratios, student choices regarding major fields of study, requirements for academic space and student housing, and so forth.

The first and second objectives have been met. An easy-to-use interactive information retrieval and analysis system—also called FIND—is available through terminals connected to the Dartmouth Time-Sharing System. The FIND system has been operational for a year and is used by personnel in the offices of the President, the Budget Officer, Personnel Administration, Alumni Affairs, and Student Affairs.

Significant progress has been made toward the third goal stated above—namely, toward a capability for developing institutional models. The FIND system can now also operate as a program-driven data retrieval and analysis system, controllable by programs written by the user in the BASIC language. Thus programs can be written which use the full computing facilities of the Dartmouth Time-Sharing System.

The purpose of this report is to discuss some of these applications of the FIND system and to indicate directions that further development will be taking.

THE FIND SYSTEM

Project FIND began to take shape in December 1971, and was officially launched in March 1972. In that month Dartmouth President John G. Kemeny met with a group of college officers and consultants to discuss the purposes, a timetable, and a tentative design for an information retrieval and modeling system.

The first version of the information retrieval system was completed by June 1972, the bulk of the programming having been done by a group of undergraduate students in a course taught by President Kemeny. Two of these students continued to work on the project during the summer as members of the FIND staff, and by September 1972, the original programs had been refined, new programs had been added to allow simple statistical analyses and cross-tabulations, several data bases covering faculty and administrative officers had been brought into existence in a format specifically designed for Project FIND, and the first modifications in the FIND system that would make it accessible to users' programs (for modeling purposes) were completed.

One of the fortuitous aspects of the modeling capability was that users across the campus were able to write programs in BASIC to manipulate their own data available through FIND. Some of these user programs were subsequently studied by the Project FIND staff and incorporated into the official FIND system for everyone to use. Thus any member of the Dartmouth community is a potential contributor to the FIND system, and the evolution of the system in directions needed by its users is ensured.

A great deal of effort was devoted to making the FIND system easy to use, and a beginning user needs to know only a few commands in order to access FIND. The most important of these commands is RETRIEVE, which specifies the general area of the user's inquiry. For example, the command

```
RETRIEVE FACULTY, SURNAME, DEPT, RANK, SEX
```

indicates that the user is interested in the faculty data base and wishes to examine the surname, departmental affiliation, rank and sex of its members. These four qualifiers are called

```
DARTMOUTH TIME-SHARING
LINE 231, ON AT 12:34 12 DEC 73, 119 USERS
USER NUMBER --123456,[redacted]
NEW OR OLD--OLD DPCLIB***:FIND
READY

RUN

FIND      (COMPILED)  12 DEC 73  12:38


FIND HERE!

?RET ADMIN, SURNAME, INITIALS, GRADE, SEX, DEPT, HIREDATE
*RET:   6 ATTRIBUTES RETRIEVED FOR 271 ENTITIES
        LAST MODIFIED 12/07/73
DONE

?SELECT DEPT = ADM
*SEL:   14 ENTITIES SELECTED
DONE

?SORT -HIREDATE
DONE

?FORMAT  W51
DONE

?PRINT INITIALS, GRADE, SEX, DEPT, HIREDATE

INITIALS  GRADE  SEX  DEPT   HIREDATE

   P        1     M   ADM    730906
   J        1     F   ADM    730901
   G        1     F   ADM    730821
   W        1     M   ADM    730814
   J        1     M   ADM    730430
   P        1     F   ADM    720901
   L        4     F   ADM    720201
   L        2     F   ADM    710819
   W        5     M   ADM    690701
   T        3     M   ADM    680901
   G        5     M   ADM    640915
   T        6     M   ADM    631001
   J        4     M   ADM    620901
   T        7     M   ADM    360701

DONE
```

Figure 1—A sample session with FIND

attributes; a typical data base may have over one hundred attributes for almost a thousand members. The attributes associated with the RETRIEVE command are collectively known as the working data base. A user's working data base is usually very much smaller than the full data base.

If the user wishes to focus his inquiry on a subset of the faculty, he may use the SELECT command, which restricts the system's attention to members who meet the user's specified criteria. For example, the command

SELECT DEPT = "HIST"

would narrow the scope of the inquiry to include only members of the History department.

The SORT command may be used to reorder the working data base elements in either ascending or descending alphabetical sequence based on specified attributes. For example, the command

SORT RANK

will sort the previously retrieved data elements in ascending order by academic rank. To obtain a descending sequence, merely precede the attribute name by a hyphen, as in

SORT -RANK, -SURNAME

which sorts the working data base in reverse alphabetical sequence within descending order of rank. The SORT command does not cause a physical rearrangement of working data entities but merely supplies a list of pointers which indicates the order in which entities are to be processed by subsequent commands (e.g., PRINT).

The PRINT command is used to display the working data base after it has been reduced and rearranged by SELECT, SORT and other commands. The example in Figure 1 shows how to obtain a list of Admissions Office administrative personnel, in descending order by appointment date. Note the use of a FORMAT command to obtain a printout which would fit on that page.

One of the most useful commands is DEFINE, which permits a user to generate new attributes based on combinations of existing attributes and constants. Attributes created through DEFINE are accessible to all other procedures in the FIND system, including DEFINE itself. The DEFINE language is modeled very closely on the Basic available on DTSS, with the exception that attribute names may be used in place of the one or two character variables permitted in Basic. The following example illustrates the use of DEFINE to compute retirement years for Dartmouth faculty members. Normal retirement is at age 65 except that people born after September retire a year later.

```
DEFINE RETIRE
10  LET RETIRE = BRTHYEAR + 65
20  IF BRTHMNTH < 10, THEN 40
30  LET RETIRE = RETIRE + 1
40  END
```

Statistical functions are available through commands such as XTAB, which performs cross-tabulations. For example, to obtain a breakdown of rank versus sex for the previously selected members of the faculty, the command

```
                                 ↓—(standard groupings)
XTAB  RANK, PTS, SEX, STD, %
            ↑————————————————— (starting points)
```

would enable the user to specify up to eight ranges for the

```
RETRIEVE  FACULTY, SURNAME, SEX, RANK
*RET:   3 ATTRIBUTES RETRIEVED FOR 523 ENTITIES,
        LAST MODIFIED 12/05/73
DONE


?XTAB  RANK,PTS,SEX,STD,%
*XTA:  SPECIFY INITIAL POINTS FOR RANK
? 1,2,3,4


SEX          M         F

RANK
1          96.4%      3.6%
2          89.1%     10.9%
3          81.4%     18.7%
4          73.5%     26.5%

 523 IN SAMPLE.   67 EXCLUDED.
```

Figure 2—Sample frequency table

attribute rank, after which FIND would display percentage figures for each sex within the specified ranges. The example in Figure 2 shows a retrieval followed by a cross-tabulation. Appendix I contains a list of the available FIND commands.

THE FIND DATA BASE STRUCTURE

A *data base* is a structured collection of information. For example, data bases may contain student grades, employee information, or a budget. In general a data base may be considered as a two-dimensional array. Each row in the data base represents one *entity*, e.g., a student, a faculty member, or a line of the budget. Each column of the data base represents some *attribute* of the entities, e.g., age, sex, name, rank, or account type. The value of an attribute associated with a particular entity is called an *item*. Items may be either numeric or string valued. Figure 3 shows the relationship between entities, attributes and items.

All attributes and all entities associated with a given data base comprise the *permanent data base*. Each permanent data base has a *directory* which describes how the information in the data base is stored. The directory includes such elements as attribute name, file name, protection level, attribute type, etc. The relationship between a permanent data base and its directory is shown in Figure 4.

In most instances, only some of the attributes contained in a permanent data base are interesting or useful for the particular problem at hand. Therefore, the RETRIEVE procedure has responsibility for accessing the data base, decoding it, and extracting those attributes which are of interest. This collection of data is then called the working data base. The relationship between permanent and *working data base* is shown in Figure 5. Most FIND commands operate only on the working data base.

INSTITUTIONALIZATION OF FIND

The first data bases that were accessible to the FIND system were constructed on an ad hoc basis for experimentation. Thus, although they served this purpose well, they could not be depended upon for up-to-date accuracy. They were not yet the "official" data bases of the college.

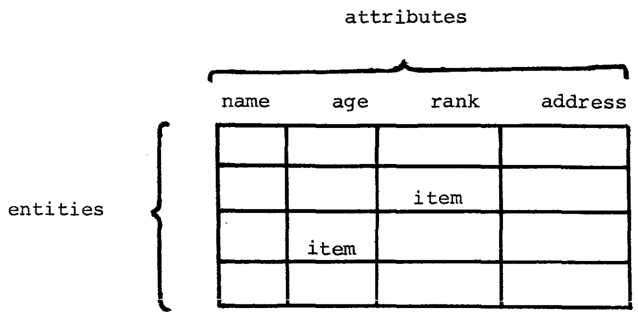


Figure 3—FIND entities, attributes and items

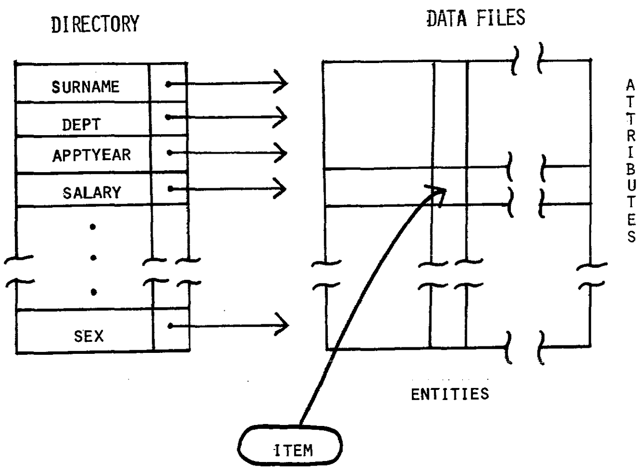


Figure 4—Directory-data base relationship

Throughout the academic year 1972-73 a major effort was directed to four problems that required solution before the institution could "turn over" its data keeping functions to Project FIND.

1. *Updating of files.* An effective and simple capability for creating new data files and for keeping old ones up-to-date had to be developed. Ultimately, the accuracy of FIND's data could be guaranteed only if the many officers of the college responsible for *generating* such data could also be assigned the responsibility for *updating* the FIND files. Project FIND's data bases had to become the primary and

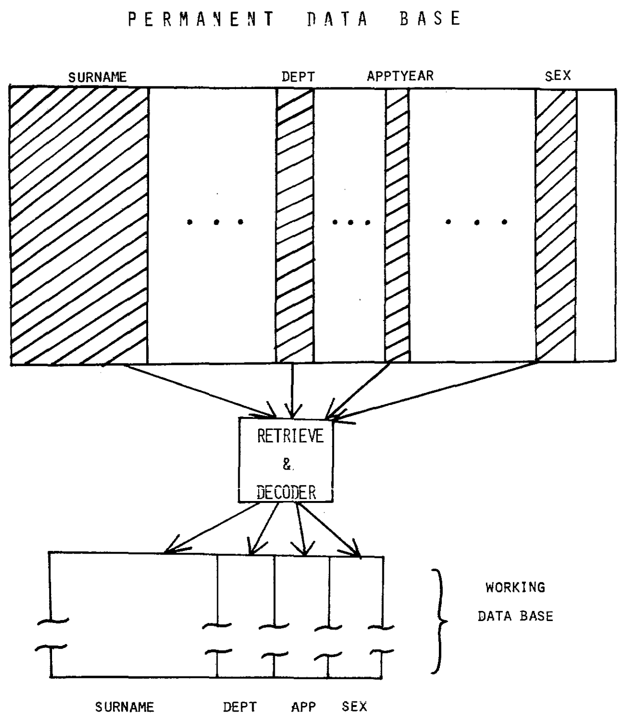


Figure 5—Relationship between permanent and working data bases

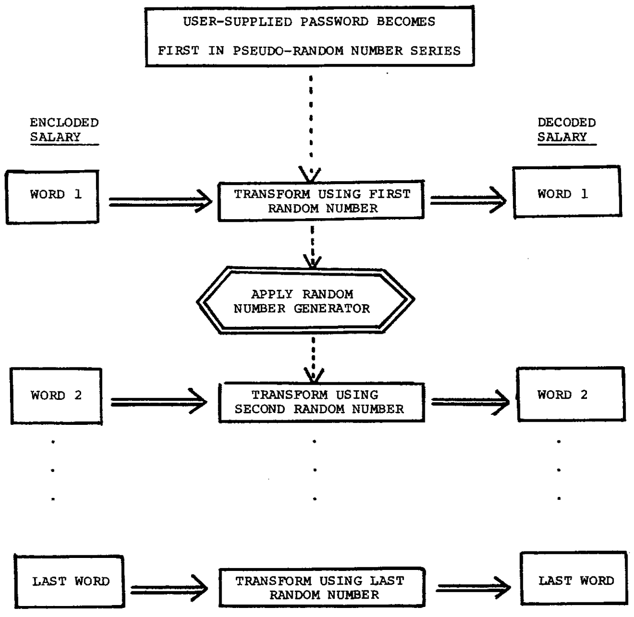

USER-SUPPLIED PASSWORD BECOMES FIRST IN PSEUDO-RANDOM NUMBER SERIES
ENCLODED SALARY
DECODED SALARY
WORD 1
TRANSFORM USING FIRST RANDOM NUMBER
WORD 1
APPLY RANDOM NUMBER GENERATOR
WORD 2
TRANSFORM USING SECOND RANDOM NUMBER
WORD 2
LAST WORD
TRANSFORM USING LAST RANDOM NUMBER
LAST WORD

Figure 6—FIND enciphering technique

"official" repository of the institution's data, not merely copies of it.

By the summer of 1973 the required capability for updating files was in existence. From any computer terminal of the Dartmouth Time-Sharing System, a user can enter the FIND system and issue the command "UPDATE". He can then effect a succession of changes in any data files under his control by merely typing the additions, deletions, and modifications directly to the computer. Alternatively he can accumulate a log within the computer of changes to be made to an official institutional data base, and he can have such changes effected automatically by making an authorizing call to the Data Processing Center.

Given the existence of the UPDATE system, Dartmouth assigned official responsibility and accountability to the Office of Personnel Administration for the maintenance of all FIND data bases relating to employees of the college. As of October 1, 1973, the FIND personnel files are the official files of the college and are kept up-to-date on a day-to-day basis. The FIND files supersede all of the previous ad hoc files that were often inaccurate.

2. *Selection of data elements.* It was necessary to make a major study of what data is needed by each office or part of the college to determine who generates and uses such data, and to learn how data flows from one part of the college to another. The Office of Institutional Research and Analysis made such studies in the areas of students and personnel. The studies produced an inventory of all data needs, identified the redundancies in the collection of data, and pointed out inconsistencies in the definition and use of data by different offices. The recently established *Personnel Data Base* was made possible as a result of these studies. And the *Student Data Base,* currently under construction, relies heavily on the inventory study.

3. *Protection of information (confidentiality).* It was necessary to develop the technical capability for protecting confidential and sensitive data from unauthorized and unintended use. By the late fall of 1972, a highly ingenious security system was implemented, using a technique of enciphering sensitive data. The user effects the enciphering through a password that he himself gives and which is retained nowhere in the computer system.

The enciphering technique works as follows. First, consider a sensitive attribute, such as salary, which is composed of a long series of n-digit quantities. (The data are stored on disc in an inverted fashion, with all salary values together in the file.) If there are 300 entities in the data base and salaries are five digits, then the salary attribute contains 1500 digits. These digits occupy a certain number of computer words; the exact number depends on the particular hardware and data packing techniques used. In the FIND system, 1500 digits occupy 167 36-bit words.

Now consider the password supplied by the user, which in FIND is an eight-character (72-bit) quantity. This password becomes the first in a series of 167 pseudo-random numbers, each with 72 bits. Thirty-six bits (one word) are extracted from each pseudo-random number and used to decode a word from the salary attribute. A schematic representation of this procedure is shown in Figure 6.

The specific operation applied to the encoded word by the thirty-six bit random number is an "exclusive or". Because this operation is a reversible transformation, exactly the same procedure (and the same 72-bit password) is used to *encode* the data in the process of setting up the FIND data base.

So effective is the enciphering technique, that even a minute difference in the password given will result in totally unrecognizable gibberish. If a user loses his own password, not even the FIND staff can help him out. The only possibility is to re-create the data base.

4. *Data Base Accessibility.* It has become necessary to address the question of *who* is entitled to access data in the

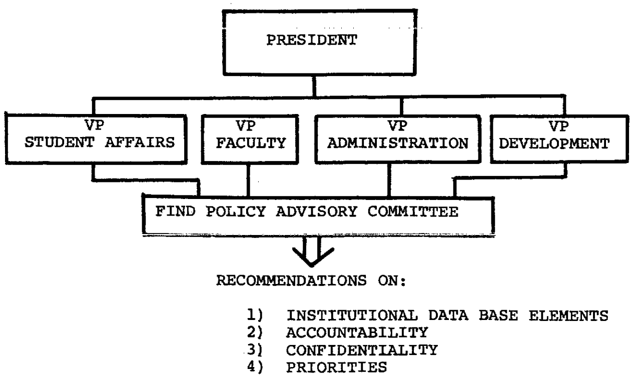


Figure 7—Project FIND organizational structure

FIND files. This is, of course, a matter of college policy, and it is being resolved through extensive consultation with college officers responsible for collecting data. The final policies adopted must accommodate the rights of individuals who voluntarily provide information to the institution, yet the purposes of Project FIND as a planning tool must not be unduly restricted.

An institution-wide advisory committee for Project FIND has worked on such matters throughout the past year under the chairmanship of Vice President Donald Kreider. This group approved in principle last May a set of recommendations from the Office of Institutional Research and Analysis on which data should be considered *institutional* data (as opposed to "private" data relevant only to one office). The committee also approved in principle a set of guidelines on *Confidentiality*, *Accessibility*, and *Accountability*. Documents on all such matters considered by the FIND staff and the several advisory committees are kept on file and should be useful to other institutions working their way through the maze of technical, political, and organizational problems entailed in the development of management information systems. This aspect of the Project FIND organization is shown in Figure 7.

REPRESENTATIVE DATA BASE

The budget data base

The budget of the institution will constitute a major data base under FIND that will tie together all the other data bases—personnel, students, space, etc.—in the modeling process. Substantial progress has been made toward creating this data base.

Appropriate identifiers for the more than 20,000 college budget accounts have been agreed upon. Attributes that provide both current and historical information about each account have been recast into FIND format. Attributes that relate each account to the college's administrative structure and to other data bases (e.g., personnel and space) have been included. And an attribute has been included that achieves a crossover from Dartmouth's organizational budget format to the functional categories proposed by NCHEMS (National Center for Higher Education Management). (Dartmouth is a participating member of the NCHEMS project.)

A test budget data base has been loaded into the FIND System, and FIND system programs have been used to manipulate this data and to identify unanticipated problems. For example, problems with the method of encoding the data were detected and corrected. Additional software requirements of the FIND system needed for efficient handling of very large data bases were spelled out and recommended to the FIND staff; these are currently being developed. And, programs for producing the variety of budget reports needed by operating managers were written and debugged.

The full institutional budget has now been recast in FIND format and has been successfully loaded experimentally. Declaring it to be the "official" budget data base awaits only the required addition to the FIND system of two technical features: (i) a capability of retrieving efficiently a part of a data base as opposed to the entire data base, and (ii) a catalog feature that will enable each college officer to retrieve just his part of the budget without access to the full budget.

The inclusion of a NCHEMS attribute in the budget data base makes it immediately possible, under the FIND system, to recast the Dartmouth budget in a form that permits comparison with other institutions that cooperate with NCHEMS. The next crucial step in the long-range development of FIND is in producing further attributes that permit the reformulation of the budget along functional lines defined especially for Dartmouth. A major effort has begun on the development of alternative budget models, and these are expected to be operational by late 1974.

Other data base

There are several uses of Project FIND that are spontaneous and not part of the main system development. Three examples serve to illustrate how the accessibility of the system lends itself to innovation on the part of the college community:

(i) The Dartmouth College Art Gallery maintains an inventory and monitors the distribution of its approximately 14,000 art objects through the FIND system. The file of art objects is updated by the Art Gallery staff rather than by the Data Processing Center.

(ii) A complete inventory of faculty involvement in committee and student advisory capacities is maintained in a FIND data file and is updated continuously by secretarial staff in the Dean of Faculty's Office. This information was kept by the Registrar's Office for several years utilizing computer programs accessible to exactly one person on the campus. In September 1973, the information was translated into the FIND system and became immediately available to all faculty committees and department chairmen through their own computer terminals.

(iii) During the summer of 1973, a request from the President for historical information on college investments resulted in the creation of a FIND data base as the best vehicle for transmitting the required information to the President. Although a small data base, containing 21 different attributes for 62 different entities, its true value lay in not restricting the President's knowledge to answers to explicitly formulated questions. Instead, he will be in a position to experiment with the investments data base and to quickly answer his original questions plus many more that can only be asked later.

Appendix II contains a partial list of available data bases.

APPENDIX I—FIND SYSTEM COMMANDS

Elementary Commands

ATTRIBUTES	List attributes in any data base
EXPLAIN	Explain a command
RETRIEVE	Sets up working data base for a session with FIND
SELECT	Extracts specified subset of working data base
SORT	Re-orders working data base by specified criteria
PRINT	Displays the working data base at the terminal
STATISTICS	Computes statistical measures
CANCEL	Halts a command
STOP	Terminates a session with FIND

Advanced Commands

CORRELATION	Print correlation between two or more attributes
DEFINE	Composes new attributes based on old ones
EXECUTE	Performs FIND commands stored in a file
FIT	Provides linear and exponential fit routines
FORMAT	Allows flexible formatting of output
OUTPUT	Allows output to a saved file
RETRIEVE	Additional information on use of the RETRIEVE command
SORT	Additional information on use of the SORT command
XTAB	Provides frequencies and cross-tabulations

Miscellaneous Commands

COUNT	Count attributes and entities
LABEL	Insert textual information into FIND output
LOAD	Retrieve a previously saved working data base
REDUCE	Permanently shrink a working data base
RENAME	Change the name of an attribute in the working data base
RESTORE	Restore the working data base to original state
SAVE	Save a copy of the working data base
SCRATCH	Destroy a portion of the working data base
TIME	Print running time

Data Base Maintenance Commands

UPDATE	Initiates a data base update session
IDENT	Specifies a list of identifying attributes
LOG	Specifies a file in which to record changes
EXIT	Terminates a session with UPDATE
ALTER	Specifies data items for immediate modification
MODIFY	Specifies a list of attributes for deferred modification
ADD	Creates a new entity
DROP	Deletes an entity from the data base

APPENDIX II—AVAILABLE DATA BASES (DECEMBER 1973)

$$\left.\begin{array}{l}\bullet\ \text{ADMINISTRATIVE}\\ \bullet\ \text{FACULTY}\\ \bullet\ \text{STAFF}\end{array}\right\}\ \text{PERSONNEL}$$

- DEPARTMENTAL BUDGETS & EXPENDITURES
- STUDENT COURSES
- PHYSICAL FACILITIES
- ART GALLERY COLLECTION
- ALUMNI (DEVELOPMENT OFFICE)
- ENDOWMENT FUNDS

Design considerations for microprogramming languages*

by GREGORY R. LLOYD and ANDRIES VAN DAM

Brown University
Providence, Rhode Island

INTRODUCTION

Historically, microprograms have been developed using tools which are appropriate to logic designers (block diagrams, register transfer languages), or systems programmers (microcode assemblers).[1] With the growth of user microprogramming, and the increased demands placed upon computer manufacturers for firmware support, improved tools and techniques have been suggested. In particular, microprogram compilers, i.e., compilers which translate high level source statements into sequences of microprogram control words, have been proposed and implemented.[2,3] The larger issue to be faced is the nebulous task of supporting the needs of a community which includes:

- Manufacturers developing one particular computer (target machine) using microprogramming as a design and implementation technique.
- Manufacturers developing a computer *system*, integrating hardware, operating systems, and programming language support, utilizing microprogramming to tie system functions together.
- Users developing their own target machines tailored to applications such as computer graphics, signal processing, or interpretation of higher level languages.
- Users or manufacturers who utilize hardware which is microprogrammable to directly solve one or more specific problems, without formally defining a target machine (i.e., simply programming the hardware, or host machine).

All groups may be able to employ the same particular host machine (a microprogrammable computer), but their needs are shaped by different viewpoints. The remainder of this paper attempts to focus on the basic issue of problem solving methodologies, particularly, what can be done to make the task of the microprogrammer less difficult, regardless of the application.

Probably the major consideration in the design of a microprogramming language is a pragmatic issue—efficiency of generated code. The second section briefly describes some of the problems involved in compiler design. Language design (third section) is constrained by practical limitations on the complexity of the compilation process. Within these limits, determined by machine architecture, it may be possible to design a useful higher level microprogramming language. A specific example of a tailored microprogramming language is presented in the last section.

MICROPROGRAM COMPILERS

General issues

There are at least two distinct approaches to providing higher level support for a user microprogrammable computer. One is definition of a hospitable target machine over the host machine's basic hardware (and possibly a systems language which generates code for the target machine). It is possible to define high level primitives in the target machine's repertoire which closely match the facilities provided in a higher level language, including flow of control, expression evaluation (for complex mathematical operations), data manipulation, etc. A number of papers presented at the SIGPLAN/SIGMICRO Interface Meeting (May, 1973) treat the topic of instruction set design in some detail.[4–7] A typical target machine could include general arithmetic, logical, and executive primitives in an efficient manner (particularly given some hardware support to speed instruction decoding[8]).

The other approach is to make the microprogramming environment *itself* reasonably hospitable by providing higher level support for the direct generation of microcode. This also has received some attention in the current literature.[2,3,9,10,11] Defining the phrase "high level language" is one immediate problem. Within this paper, a high level language is defined as one which has at least the following set of features:

1. Symbolic user variables (allocated by the compiler),
2. Ability to evaluate arbitrary arithmetic or logical expressions,
3. Flow of control statements beyond simple (conditional and unconditional) GOTO, SKIP, Branch and Link.

It should be stated that the target machine approach and the microprogram compiler approach are not mutually exclusive: language and target machine design can proceed

* This research was supported by U.S. Naval Research Laboratory Contract N00014-67-A-0191-0027.

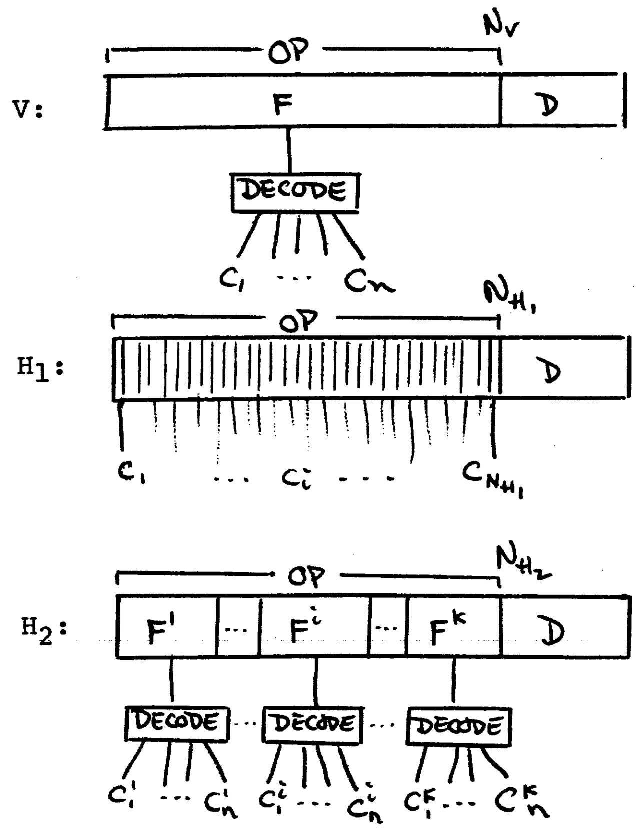


Figure 1—Microinstruction formats

simultaneously. For example, the Microdata 32/S computer and the MPL language were designed in parallel, reflecting language constructs directly into target machine operations, and *vice versa*.[12] The end result is a PL/I subset which is the target machine language in much the same sense that ALGOL is the machine language of the B5500. In another sense, a higher level language for the host machine can be considered an abstraction of the host hardware, shielding the programmer from details (such as loading a shift-amount register), but constraining him to those functions which are implemented at a hardware level.

Host machine microprogram compilers

The general advantages of programming in a higher level language apply when the compiler happens to generate microcode—namely, increased programmer productivity, self-documentation, improved maintainability, ease of training, etc. In addition, an application running in direct microcode could gain advantages in efficiency and security over a similar software based system. A language can be designed on a number of levels ranging from problem oriented languages through "general purpose" languages to machine dependent languages. Depending upon the design level of the language, the programmer can be fully isolated from the hardware or can be presented with a set of facilities which allow him to interact at a register level (a "CODE . . . ENDCODE" statement, which allows insertion of assembler or machine language instructions in-line[13]). The concept of language extensibility can be used to tailor a base language to a specific set of applications by, for example, enriching the run time environment with operating system primitives when appropriate.

The major disadvantage of microprogram compilers is the conjectural nature of statements on efficiency of compiler produced code. Microprogramming has traditionally been viewed as a fairly esoteric discipline, requiring complete knowledge of the host architecture and a willingness to spend considerable time tuning programs to (simultaneously) conserve control storage and achieve maximum performance. If the quality of compiler produced code is significantly worse than that produced by hand, the ergonomic advantages of coding in a higher level language may be outweighed. Hopefully, the efficiency requirement can be recognized as a major influence on language design, and feature selection can be guided by (1) the functions which can be performed efficiently in hardware and (2) recognition of constructs which can be compiled to efficient code. Optimization of code produced for some classes of microprogrammable machines is inherently difficult, and, except for work by Ramamoorthy and Tirrell, relatively unexplored in the literature.[3,2,14]

The following section treats some of the problems in compiler design which relate to the decision to generate microcode directly from a higher level language.

Problems in compilation to microcode

Microinstruction formats

Three possible microinstruction formats are illustrated in Figure 1. The first (labelled V) is typical of the vertically encoded format. One microinstruction (MI) consists of a control field F, and a data part D (which might specify register number(s), literal data, a branch address, etc.). The N_v bits which comprise the operator part (OP) are fed through a decoding network, controlling up to 2^{N_v} different lines, c_i. Each control line determines a micro-operation (MO) which might, for example, perform an arithmetic operation, transfer the contents of one internal register to another, initiate a memory read, etc. For format V, one micro-operation is specified in one microinstruction.

The format labelled H_1 is an example of a horizontal direct control format.[1] Each bit in the operator part controls one of N_{H1} lines. In general, not all of the OP bit patterns are legal, since some micro-operations cannot be executed in the same machine cycle (because of conflicting register or function utilization). The number of micro-operations which could be specified per microinstruction is N_{H1}, the number of control lines.

The third format (H_2) represents a horizontal field-encoded microinstruction. The OP is split into several fields, each of which is decoded to control one of a set of (related) control lines. Again, not all values of the OP may be legal. It is also possible for one field (say, F^1) to determine the layout and interpretation of the remaining fields. The number of micro-operations specified per microinstruction is K, the number of encoded fields.

In practice, horizontal machines may employ a mixture of direct and encoded control fields, and "vertical" machines may have more than a single encoded field. For the sake of discussion, each field for H_2, each control bit in H_1, and the single field in V will be said to specify the execution of one micro-operation. The set of operations in a horizontal machine will be described as *independent* if, during the execution of each machine cycle, any bit pattern of the OP designates the execution of N_{H1} (for H_1), or K (for H_2) parallel micro-operations (the interpretation of the data part is also assumed to be independent of any micro-operations specified). Again, the concept of complete independence of micro-operations is unrealistic, but the degree to which the independence is constrained is an important concept in the following section.

Compilation models

In the present context, compilation is the process of translating statements expressed in a source language L into a sequence of microinstructions. One common compiler design divides this mapping into two separate translation phases. In the first, syntactic recognition and semantic analysis produce an intermediate form of the source, which will be referred to as a sequence of language primitives (lp's). Typical primitives include arithmetic and logical operations, register loads and stores, condition testing and branching, etc. In the second mapping, lp's are transformed into a sequence of microinstructions which may be executed directly.

In symbolic form (quadruples, triples, duos[15]), lp's might resemble assembly language statements for a sophisticated target machine, with instruction addresses replaced by internal linkages and operand addresses replaced by symbol table pointers. The mapping from lp's to microinstructions could be one-to-one for simple operations, or one-to-many, for operations such as procedure calls (or arithmetic operations, where a 16-bit adder might be used for 32-bit arithmetic).

Figure 2 illustrates three models for compilation, indicating the two mappings described so far in the boxes labelled "translation" and "generation". The first (V) is the compilation process for a pure vertical machine. The generation of microinstructions is a process of selecting a sequence of (legal) OP bit patterns and data parts (the same as the code generation phase of a normal compiler). The ordering of the microinstructions may be modified, subject to *data* dependent constraints (input/output relations[14]).

The second and third diagrams in Figure 2 represent compilation models for horizontal machines. The HI diagram represents the compilation process for a machine with *totally* independent micro-operations. The generation phase produces a stream of micro-operations. Since only data dependencies exist, the micro-operations may be reordered and optimized, much as the microinstructions for case V. The composition phase combines MO's into microinstructions, utilizing data dependency information from the translation and generation phases. The complexity of the composition step is determined by the number of micro-operations within each horizontal control word.

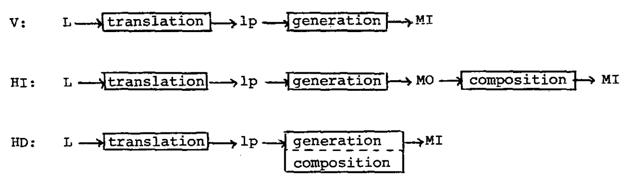


Figure 2—Compilation models

As MO constraints are added, the situation approaches that illustrated by HD, where the generation and composition phases are inextricably bound, i.e., the rules governing the composition of MO's are so complex and interrelated that it is no longer useful to consider the generation of a stream of independent MO's. The emphasis shifts back to determining a sequence of legal OP bit patterns which perform some composite action. This analysis may be performed statically, considering the OP as one *large* operation code field, or dynamically, by searching for the required bit pattern on an MI-by-MI basis. In either case, the probability of determining a sequence of microinstructions which utilize all available resources most efficiently is low (or the cost is correspondingly high).

In any realistic case, MO constraints for horizontal machines place the compilation complexity somewhere between HI and HD. Two distinct phases may still exist, but the complexity of the composition function is increased to include resource, as well as data scheduling.[14,2] It is useful to categorize the difficulty of microcode generation for a specific machine in terms of:

(1) Combinatorial complexity proportional to the number of MO's within each word.
(2) MO dependency in terms of number of shared resources which must be managed.
(3) Timing dependency, i.e. the scheduling of operations which require more than one machine cycle to complete (for example, a core reference).

Any of these factors complicate the code generation process, and their combined effect is synergistic. In these simple terms, an ideal horizontal organization for the efficient compilation of microcode would utilize field encoded control (reducing the number of MO's per word), each field managing a disjoint resource set, with no timing dependencies. To some extent, this type of design is exemplified by the MCU[16] and the AMP.[17] A different approach to providing a horizontal host machine is the QM-1,[19] where a horizontal (360 bit) nanocontrol word specifies the micro-operations to be executed within one major cycle, and an 18-bit field in the microinstruction selects a particular nanoword.

MICROPROGRAMMING LANGUAGE DESIGN

Discussion

The previous section concerned itself mainly with compilation from an unspecified language through to microcode for a variety of machines. Within this section, the inverse problem is considered, i.e., given a host machine. is there a higher level language which may be defined to ease the microprogrammer's task, without exceeding some "reasonable" efficiency bound?

Microprogramming language levels

Categorizing the levels of microprogramming language support is a relatively difficult task, due in part to some of the peculiarities of horizontal machines, as outlined in the previous section. One hierarchical description would include:

(1) Symbolic microinstruction assemblers—Straightforward field sensitive specification of control word contents.[1]
(2) Symbolic micro-operation assemblers—Each statement of the language corresponds to one microinstruction. Within a statement, micro-operations may be expressed symbolically (for example, in register transfer notation), AMIL.[19]
(3) Symbolic micro-operation languages—A sequence of MO statements is composed *by the compiler* into microinstructions.
(4) Restricted language with micro-operation statements—A mixture of MO statements and language constructs which map (one-to-many) onto MO's. Typically, higher level constructs include flow of control (IF . . . THEN . . . ELSE, conditional DO's, etc.). Composition of MO's is performed by the compiler. The level of support is analogous to that provided by PL360,[20] in that a detailed knowledge of the host machine's architecture is required. Examples are in References 2, 3 and 9.
(5) Machine dependent languages—Higher level languages with some machine dependent features, but general symbolic facilities such as expressions, data aggregates (arrays, structures). Explicit specification of MO's and host machine registers would not be necessary.[21]
(6) Machine independent language—A higher level language with features specified independently of any particular machine architecture. For example, ALGOL or an ALGOL subset.

Levels 1 and 2 provide a representation of microinstruction sequences. If any composition of MO's is required, the programmer must perform this mapping himself. Thus, detailed knowledge of all MO and timing dependencies is required. Macro extensions of these languages could provide some higher level support, but the complexity of the composition phase may practically preclude any non-local optimization of microinstruction sequences.

Levels 3 and 4 provide a composition function within the language compiler. For horizontal machines, this removes a large burden from the microprogrammer, who may concentrate on the specification of operation sequences without explicitly recognizing many of the low level dependencies which exist within the hardware. A knowledge of host machine facilities (registers, local stores) and operations is necessary, but the flavor of these languages is similar to that of normal assembly language programming. Macro extensions from these levels could be extremely useful in building type 5 support.[22]

Levels 5 and 6 may be similar in syntactic form. The distinction between the two is analogous to the differences in design philosophy between a general purpose programming language such as PL/I and a systems programming language such as LSD.* For level 5, some commonality in language features with a general purpose language is desirable, but constructs which would require a complex run time environment, or volumes of in-line code, are to be avoided (implicit data conversions, varying string operations, all but the most primitive i/o facilities, etc.). The process of feature selection as it relates to the design of a level 5 language for microprogramming is discussed in the following section.

Tailored languages

A language whose features are explicitly designed to coincide (to a large extent) with the hardware capabilities of its object machine will be referred to as a "tailored language". In the microprogramming context, this corresponds to defining a set of language primitives which map directly into micro-operations, on almost a one-for-one basis, and building a more expressive source language on this base.

For example, the set of arithmetic and logical operations supported in hardware define lp's for these binary and unary operations. At the source language level, these simple operations may be combined to form expressions. Main memory references might be defined as lp's which would be expanded to memory address register loads, and memory read/write micro-operations. This would allow the use of symbolic main store variables in the microprogramming language to be supported at the source level. Built-in functions could be included to reflect specific hardware capabilities (bit string selection, hardware instruction decoding primitives, etc.).

The extent to which this approach may be successful is determined largely by the architecture of the host machine. A complex microinstruction format (HD in the compiler model) would define a complex set of lp's, which would make translation from the source language difficult (or conversely, would define a strangely convoluted source language).

PUMPKIN—MCU SYSTEMS LANGUAGE

A tailored microprogramming language, PUMPKIN, has been designed for the Microprogrammed Control Unit (MCU) of the AN/UYK-17, currently under development by the U.S. Naval Research Laboratory.[16] After briefly describing some of the hardware characteristics of the MCU, some features of the PUMPKIN base language are discussed.

* Language for Systems Development, a systems programmer's dialect of PL/I.[13]

Descriptions of the two additional levels of the PUMPKIN language (base+operating system primitives, extensible base), as well as a proposed base language manual are contained in Reference 21. To date, only the language definition exists. Future research will investigate some of the optimization and code generation techniques which would be used to implement a PUMPKIN compiler.

MCU description

The MCU is a 16-bit processor which functions primarily as the system controller of the AN/UYK-17 Signal Processing Element. Components of the MCU include:

- Two 16-word local stores;
- Two memory data channels which (separately) address words in 32-bit buffer stores;
- A field select unit (FSU), which may select any subfield (up to 16 bits) of a 32-bit word read from buffer store;
- An arithmetic and logical unit (ALU) with 16-bit operations such as addition, subtraction, logical and, or, exclusive or, complement, equivalence, plus a shift capability;
- 16 hardwired interrupt levels;
- Communication with and control of other system components (including other MCU's) across a shared bus (Z-bus).

The MCU is a horizontal microprogrammed machine with a 64 bit wide control word composed of 16 encoded fields which determine source and destination address for parallel register transfers, ALU operations, parallel buffer store reads and writes, conditional microinstruction sequencing, etc.

The firmware development support for the MCU is a symbolic micro-operation assembler, AMIL,[19] which includes a syntax macro capability.[23] Programming the MCU requires careful consideration of operations which may proceed in parallel within one 150 nanosecond cycle.

PUMPKIN base language features

Excursions from the tailored concept (e.g., for the MCU, allowing stack operations) should be carefully examined in terms of (1) utility, (2) code generation possibilities, and (3) required run time overhead.

Definition of the primitives for the MCU is relatively straightforward. Arithmetic and logical operations (add, subtract, and, or, exclusive or, equivalence, complement, left shift, right shift, shift left circular, increment and decrement by one) on 16-bit operands, and substrings of 32-bit operands read from buffer store (maximum length 16 bits) are the primitive operations. Iterative loop control, interrupt generation/servicing, (one level) subroutine calls, and Z-bus communications all have hardware support. Extensions to these primitives include:

- Symbolic variables in buffer store or local store (submerging buffer address register operations and read/write code points in compiled code).
- Evaluation of arithmetic and logical expressions involving primitive operations.
- Unrestricted subroutine calls (number of levels).
- Flow of control primitives which involve generation of condition testing and branching code (IF . . . THEN . . . ELSE, DO WHILE, DO UNTIL), or address arithmetic (CASE).
- Data aggregates involving simple address calculations (one dimensional arrays, structures of bounded length—32 bits maximum).
- Based storage without dynamic storage allocation, i.e., allowing pointer qualifiers as in LSD, but restricting the use of based variables (including structures and arrays) to storage templates. Allocation of storage would be the explicit responsibility of the programmer.
- Automatic storage following a simple stack frame model to facilitate the writing of reentrant code (including interrupt service routines).
- Substring qualification on the left of assignments (i.e., setting bit substrings of words) which involves masking code, since the FSU selects bit substrings on input operations only.
- Low level interfaces with AMIL, i.e. hardware facility reservation and use, a CODE . . . ENDCODE statement.

The syntactic form of PUMPKIN is similar to that of LSD, with some limitations (no character strings, data elements of 16 bits or less), and several built in functions corresponding directly to conditions (carry out, adder overflow) which may be tested within the MCU (see Appendix for an example of a simple PUMPKIN program). Constructs within the language are designed so that the translation phase can produce a simple sequence of micro-operations which implement the specific function. The language itself was designed by considering features which are supported by hardware resources. For example, the field select unit allows definition of bit substring operations and bit string components of structures with low overhead in terms of compiled code.

Other features (such as multiply/divide operators) would not be provided in the base language, but could appear: (1) within a well-defined superset of the base language (another language level), (2) utilizing an extensible version of the base language to define new data types and operators, (3) as a set of micro-subroutines callable from base language programs.

REFERENCES

1. Husson, S. S., *Microprogramming: Principles & Practice,* Englewood Cliffs, New Jersey; Prentice-Hall, 70.
2. Ramamoorthy, C. V., M. Tabandeh, and M. Tsuchiya, "A Higher Level Language for Microprogramming," *MICRO6,* Sept. 73.
3. Tirrell, A. K., "A Study of the Application of Compiler Techniques to the Generation of Microcode," *SIGMICRO/SIGPLAN Interface Meeting,* May 73.
4. Schneider, V. B. and B. W. Wade, "A General Purpose Language for Minicomputers," *SIGMICRO/SIGPLAN Interface Meeting,* May 73.
5. Broadbent, J. K. and G. F. Coulouris, "MEMBERS—A Microprogrammed Experimental Machine with a Basic Executive for

Real-Time Systems," *SIGMICRO/SIGPLAN Interface Meeting*, May 73.
6. Harrison, M. C., "Language Oriented Instruction Sets," *SIGMICRO/SIGPLAN Interface Meeting*, May 73.
7. Broca, F. R. and R. E. Merwin, "Direct Execution Of the Intermediate Text From a High-Level Language," *SIGMICRO/SIGPLAN Interface Meeting*, May 73.
8. Rauscher, T. G., *On the Feasibility of Emulating the AN/UYK-7 Computer on the AADC Signal Processing Element*, NRL Memo 2525, Nov. 72.
9. Blomberg, L. and H. Lawson, "The Datasaab FCPU Microprogramming Language," *SIGMICRO/SIGPLAN Interface Meeting*, May 73.
10. Oestreicher, D. R., "General Purpose Microprogramming Language," *SIGMICRO/SIGPLAN Interface Meeting*, May 73.
11. Eckhouse, "A High Level Microprogramming Language (MPL)," *AFIPS Conference Proceedings*, Vol. 38.
12. *MICRODATA 32/S Computer Programming Language MPL*, Microdata Corporation, 73.
13. Bergeron, R. D., *et al.*, "Systems Programming Languages—A Survey," *Advances in Computers*, Vol. 12, 72.
14. Ramamoorthy, C. V. and R. L. Kleir, "A Survey of Techniques for Optimizing Microprograms," *MICRO3*, Oct. 70.
15. Gries, D., *Compiler Construction Techniques for Digital Computers*, New York; John Wiley & Sons, Inc. 71.
16. *AN/UYK-17 Signal Processing Element Architecture (preliminary)*, Naval Research Laboratory, June 73.
17. Barr, R. G., *et al.*, "A Research Oriented Dynamic Microprocessor," *IEEE Transactions on Computers*, Nov. 73.
18. *QM-1 Hardware Level Users Manual*, Nanodata Corporation, March 73.
19. *AN/UYK-17 Signal Processing Element Microcoding Support Software (preliminary)*, Naval Research Laboratory, June 73.
20. Wirth, N., "PL360," *JACM*, Jan. 68.
21. Lloyd, G. R., *et al.*, *MCU Microprogramming Language Study-Source Level Description*, Brown University Technical Report, Contract N00014-67-A-0191-0027, Dec. 73.
22. Dickman, B. N., "ETC—An Extendible Macro-Based Compiler," *AFIPS Conference Proceedings*, Vol. 39.
23. Leavenworth, B. M., "Syntax Macros and Extended Translation," *CACM*, Nov. 66.
24. Agrawala, A. K., and T. G. Rauscher, "The Application of Programming Language Techniques to the Design and Development of Microprogramming Languages," *MICRO6*, Sept. 73.
25. Anagnostopoulos, P. C., *et al.*, "Computer Architecture and Instruction Set Design," *AFIPS Conference Proceedings*, Vol. 42.
26. Magel, K. I., "Preliminary Overview of Extensions to Programming Languages," *Brown Univ. Computing Review*, June 72.
27. Magel, K. I., *Extension Mechanisms in LSD*, Brown Univ. Technical Report, Feb. 73.
28. Rauscher, T. G., "Towards a Specification of Syntax and Semantics for Languages for Horizontal Microprogrammed Machines," *SIGMICRO/SIGPLAN Interface Meeting*, May 73.

APPENDIX

The following is a example of a simple PUMPKIN program. It is presented in the interests of providing some idea of the form of the language. For a formal description of the syntax, see Reference 21.

```
PROC PARSERX (RX);
        DCL 1 RX DWORD, "32 BIT RX INSTRUCTION"
            2 OPCODE BIT(8), "OPERATION CODE"
            2 REG1 BIT(4), "SOURCE/TARGET REGISTER"
            2 STORAGE_ADDRESS BIT(20),
                3 INDEX BIT(4), "INDEX REG"
                2 BASE BIT(4), "BASE REG"
                2 DISPLACEMENT BIT(12);
        DCL PARSED_OPCODE WORD IN LSB (8); "LOCAL STORE B, LOC 8"
        DCL REGNO WORD IN LSB(9);
        DCL EFFEC_ADDRH WORD IN LSB(10);
        DCL EFFEC_ADDRL WORD IN LSB(11);
        "SIMULATED GENERAL PURPOSE REGISTERS"
        DCL 1 GPR (0:15) DWORD IN BSM1(100), "BUFFER STORE"
            2 HIGH WORD,
            2 LOW WORD;
    "THIS ROUTINE IS PASSED A /360 RX TYPE INSTRUCTION AS A PARAMETER, STORES THE OPCODE
AND REGISTER OPERAND IN LOCAL STORE, AND CALCULATES THE EFFECTIVE ADDRESS OF THE STOR-
AGE OPERAND (BASE REGISTER + INDEX REGISTER + 12 BIT DISPLACEMENT). IT THEN CALLS EXECRX
TO EXECUTE THE INSTRUCTION."
        PARSED_OPCODE ← OPCODE; "SET OPCODE IN LSB"
        REGNO ← REG1; "SET REG # IN LSB"
        EFFEC_ADDRH ← 0; "ZERO HIGH 16 BITS OF EA"
"CARRYOUT RETURNS THE VALUE OF THE ADDER CARRY WHEN EVALUATING THE EXPRESSION USED
AS AN ARGUMENT. AS A SIDE EFFECT, THE 16 BIT RESULT MAY BE ASSIGNED WITHIN THE FUNCTION."
        EFFEC_ADDRH ← CARRYOUT(EFFEC_ADDRL ←
            DISPLACEMENT+LOW(BASE)) + HIGH(BASE);
        IF INDEX ¬ =0 THEN
            IF CARRYOUT(EFFEC_ADDRL ← EFFEC_ADDRL+LOW(INDEX))
                THEN EFFEC_ADDRH ← EFFEC_ADDRH+1;
        "ZERO HIGH BYTE OF SUM (24 BIT ADDRESSING)"
        EFFEC_ADDRH ← (EFFEC_ADDRH+HIGH(INDEX)) & X'00FF';
        CALL EXECRX; "EXECUTE THE RX INSTRUCTION"
END PARSERX;                                                            "C. BERGMAN, AUG 73"
```

Controller for a flexible disk*

by RONALD G. HARRIS, JAMES E. SUSTMAN and JOHN F. McDONALD

Yale University
New Haven, Connecticut

INTRODUCTION

In late 1971 several manufacturers of mass recording media[1] began to announce initial specifications for a class of simple low-cost magnetic disks based on the IBM flexible disk initial microprogram system loader.[2] These disks were marketed for data logging and other applications where repeated read-write activity was minimal. Their extreme low cost made it possible to integrate them into inexpensive hardware configurations as a substitute for conventional disk drives. The Memorex 651 has characteristics that are typical of the available Floppy Disk drives and is selected here as the device to be interfaced.

DRIVE CHARACTERISTICS

The removable medium of the Memorex 651 is a flexible 4 mil Mylar platter about the size of a 45 rpm record, costing about $2 in large quantities. It is similar to the IBM 23FD-11. Only one surface is used for recording, the density is 3100 bits per inch on the innermost track, data encoding is Frequency Modulated (FM), and the maximum capacity of the platter is 2.5 million bits, assuming the most compact organization. The platter rotates at a speed of 375 rpm, the data rate is 250 kilobaud, and the read/write head is positioned by a simple stepping motor. There are 64 tracks on the Memorex 651 (77 track IBM compatibility is available with the Memorex 652).

A single electronics card handles all level conversion, signal amplification, and power driving appropriate to the internal operation. Communication with external TTL control logic is through line receiver/driver pairs specified by the manufacturer.[3]

The signals that concern the interface designer are illustrated in Figure 2. These signals are functionally divided into four groups which (1) control head motion, (2) sense circumferential position on the disk, (3) read the disk, and (4) write the disk. All signals to and from the disk are asserted low, that is, a (TTL) low level is interpreted as a logic 1.

* This work was partially supported by the Alfred P. Sloan Foundation.

The head motion logic receives three control signals, STEP IN, STEP OUT, and HD LD, and returns one status signal, TRACK0. Asserting HD LD (Head Load) causes a pad to press the recording surface of the platter against the head. The head is not settled until 20 milliseconds after the assertion of this signal; the interface must wait before indicating that the load operation is complete.

A 10 microsecond pulse on STEP IN or STEP OUT will move the head toward or away from the center of the disk. The head takes 10ms to move between adjacent tracks and an additional 10ms to settle on the destination track. When the head is positioned over the outermost track (track 0 by convention), it closes a switch, asserting TRACK0, which is the only data available to the interface about the radial position of the head.

The drive senses circumferential position on the disk optically, returning two signals, SECTOR H, and INDEX H. The assertion of SECTOR H indicates the beginning of a new sector on the disk. The assertion of INDEX H indicates that the next sector is sector 0. Since the drive returns only these two pulses to communicate the circumferential position of the disk, the disk must make one full revolution before the sector count in the interface can be assumed to be valid.

To write on the disk, the interface must assert WT ENB H (write enable) and then transmit the data string encoded as in Figure 3 on WT CLKS.[4] A clock signal is recorded with the data to make the read "self-clocking," as opposed to systems which have a separate timing track, so WT CLKS is simply the sum of the data and the clock. To ensure proper synchronization during a read operation, a string of 128 zeros is inserted at the beginning of every record followed by a synchronizing pattern (it need only be one bit long) in order to identify when the first word of data has been "framed." Guard zeros are appended to the end of the record to reduce noise on the last data bit. The computer or DMA channel should insert track and sector identifying information at the beginning of the data record and a checksum at the end to facilitate error detection.

The read logic in the drive separates the clock pulses from the data and gives two signals to the interface, SEP CLOCK and SEP DATA (Figure 2). The 128 bit zero header is necessary to insure correct separation; the controller ignores the header until the framing pattern is in its buffer because drive's synch circuit may miss the first bit and fail to lock onto the data immediately.

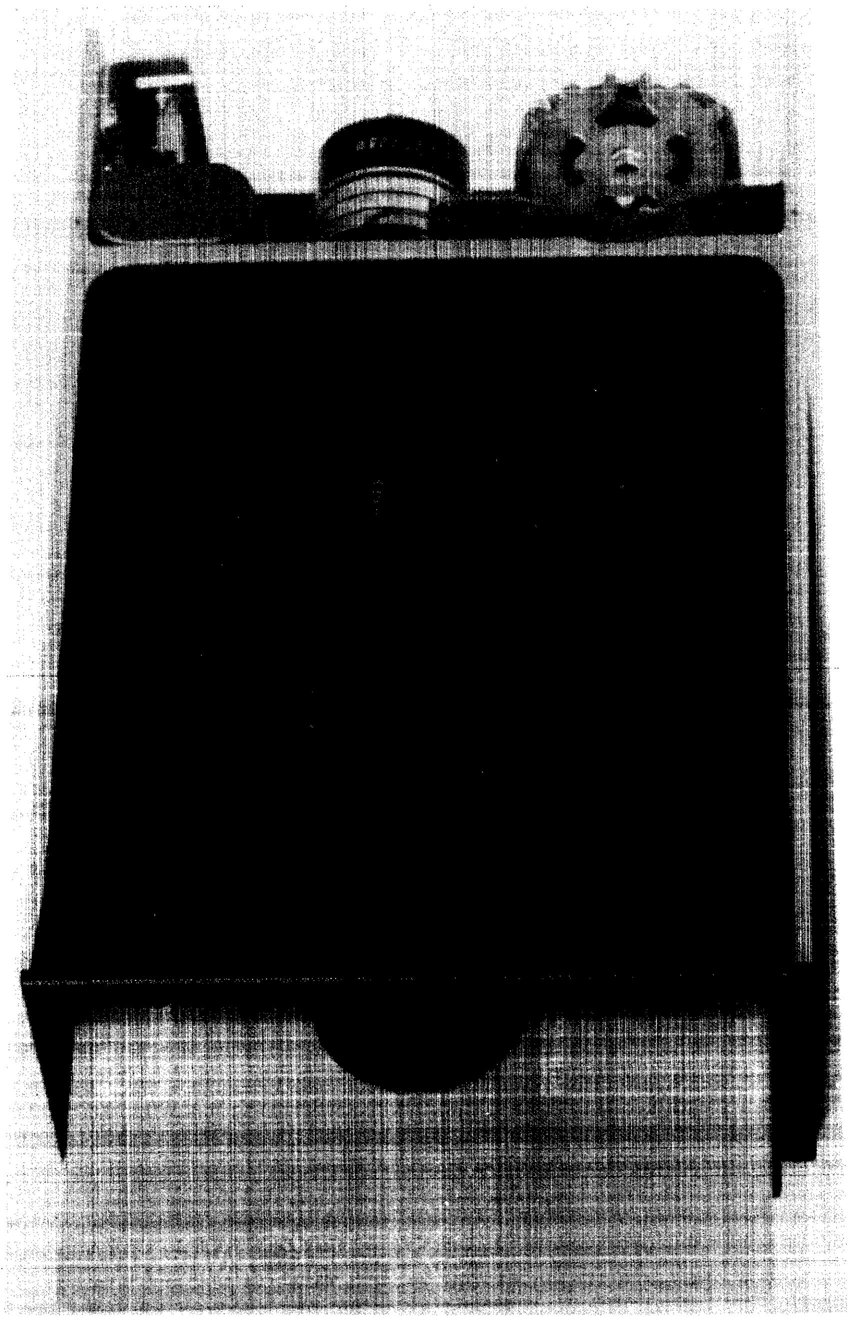


Figure 1—The Memorex 651 Drive (Reproduced with permission by Memorex Corporation, Santa Clara, California)

The drive requires three power supplies, a 5 volt, 1 amp supply for the logic, a 15 volt, 600 ma supply for the read and write amplifiers, and a 24 volt, 2 amp supply for the head position stepping motor. The power supplies are a major fraction of the cost of the interface.

DATA INTERFACE DESIGN

The design presented here is highly modular, so that it may be easily adapted to different computers or I/O architectures. (Figure 4) The modules are:

(A) Read
(B) Write
(C) Sector Match
(D) Track Position
(E) I/O

The I/O module contains the data buffers and all the logic

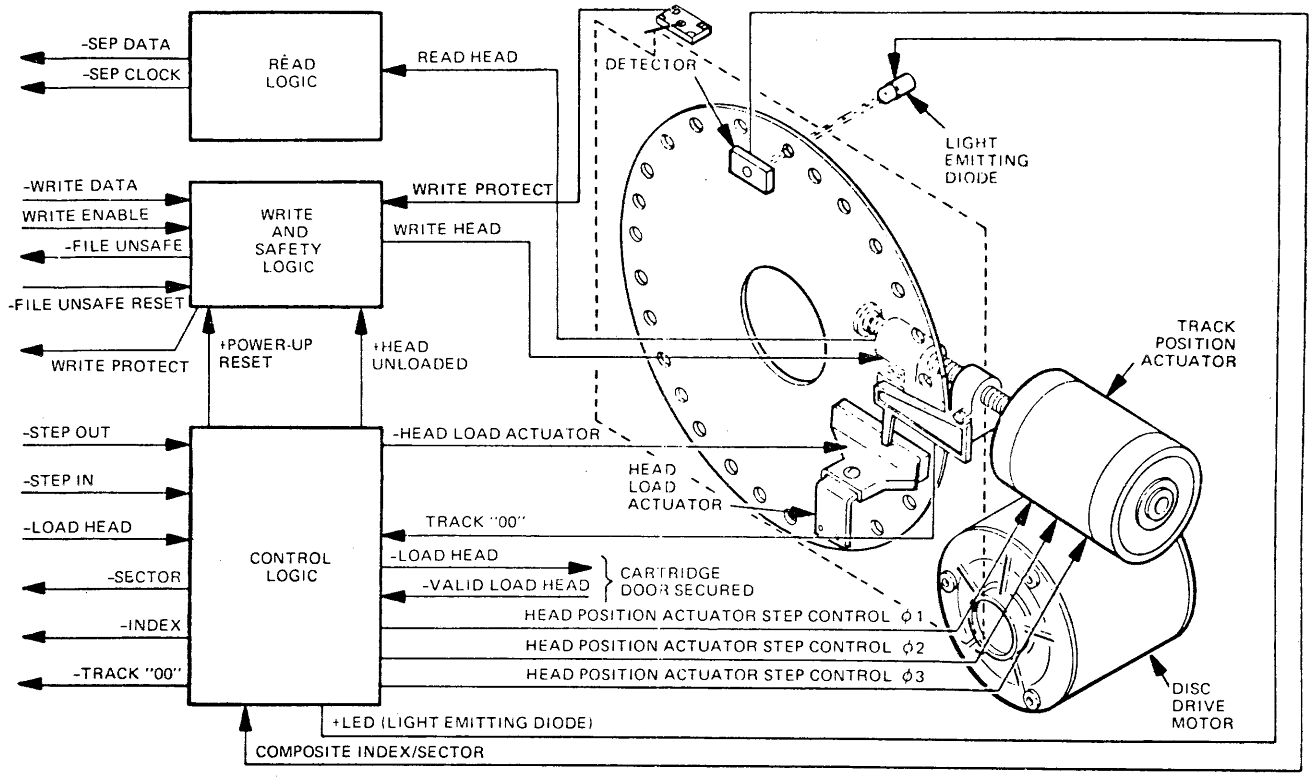


Figure 2—The Control Signal Interface for the Memorex 651 (Reproduced with permission by Memorex) as shown in the manufacturer's interface manual

to decode the control signals from the computer. We will not describe this module since its details are specific to each computer.

The Read module (Figure 5) performs the conversion of incoming data from the disk drive's serial format to 16 bit parallel format for transfer into the computer. The module also contains the synchronization logic to frame the data at the beginning of a record.

To initiate a read operation the Sector Match module issues ST RD (Start Read). The trailing (rising) edge of ST RD sets the READ flip flop. Once RD ENB H is asserted,

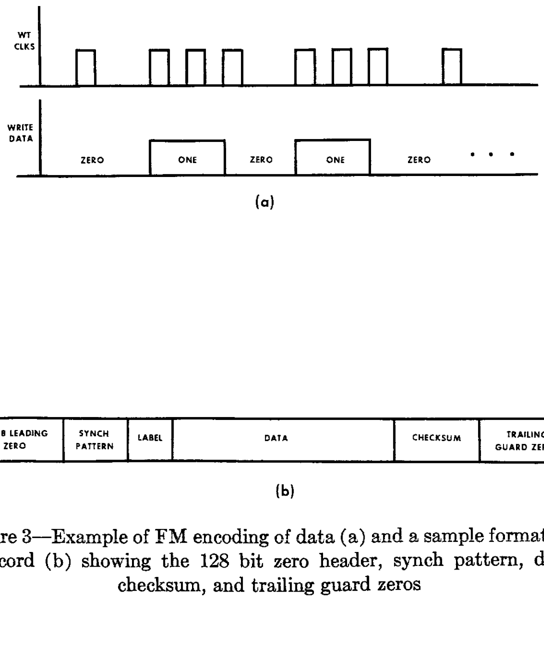


Figure 3—Example of FM encoding of data (a) and a sample format for a record (b) showing the 128 bit zero header, synch pattern, data, checksum, and trailing guard zeros

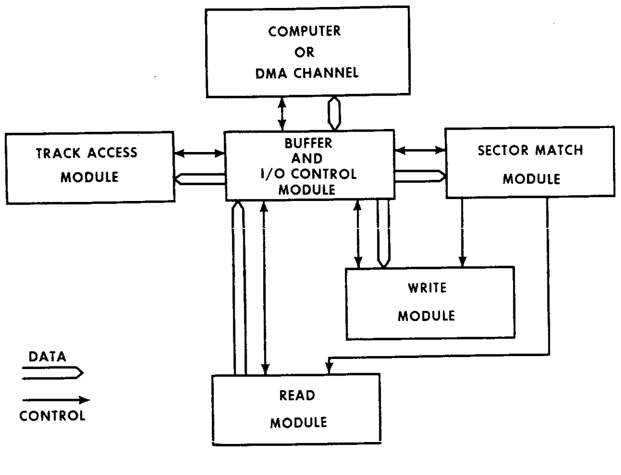


Figure 4—An overview of the organization of the controller showing data paths and control paths

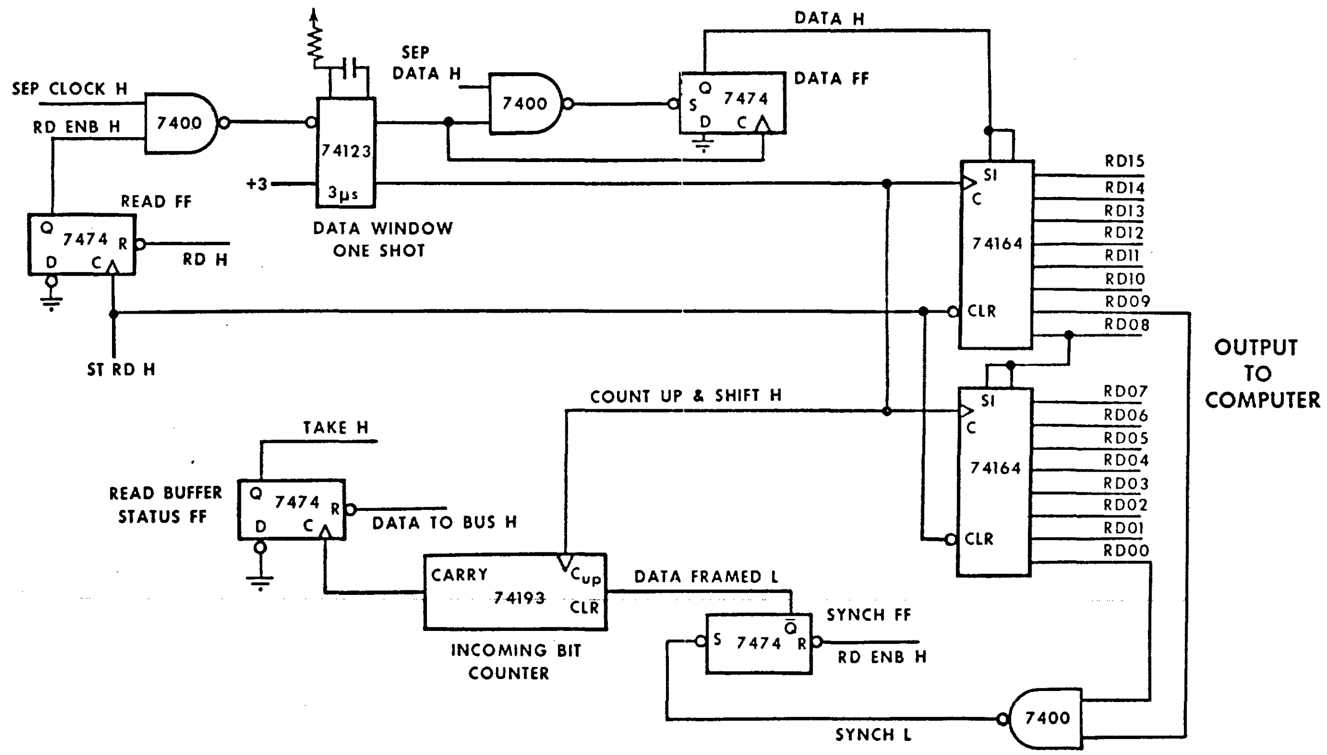


Figure 5—The Read Module Circuit (Read Buffer not shown)

SEP CLOCK is gated from the drive's synch separator to the DATA WINDOW ONE-SHOT. The leading edge of the gated SEP CLOCK initializes the DATA flip flop to zero and triggers a 3 microsecond aperture during which SEP DATA can set the DATA flip flop. If SEP DATA is asserted during this interval, the DATA flip flop will capture it. The falling edge of the DATA WINDOW ONE-SHOT clocks the data bit into the shift register and increments the INCOMING BIT COUNTER. The INCOMING BIT COUNTER asserts its carry (low) when the count is 15. The trailing (rising) edge, which occurs after the 16th bit is shifted, clocks the the READ BUFFER STATUS flip flop, asserting TAKE H. The rising edge of TAKE H should clock the contents of the shift register into a buffer and notify the computer or DMA channel to read the buffer. When the buffer has been read, DATA TO BUS must be asserted to reset the STATUS flip flop.

If the computer or DMA channel does not read the data buffer before the next carry, the new data will overwrite previously buffered data. Thus, data transfer must occur within 64 microseconds for a 250 kilobaud rate and 16 bit buffer.

The Read module synchronizes with the data at the beginning of a record by asserting RD ENB H, enabling the SYNC flip flop. Once enabled, the arrival of the synch pattern in the incoming shift register asserts DATA FRAMED L, enabling the INCOMING BIT COUNTER. The pattern used in this implementation is '1001' (octal) which is not likely to occur during the synchronizing header.

To make the data self-clocking the Write module (Figure 6) must have an internal time-base to generate the clock to which the data is added for recording. To initiate a write operation the Sector Match module issues ST WT (start write). The trailing (rising) edge of ST WT sets the WRITE flip flop, asserting WT ENB H and starting the clock, which generates three timing signals, ΦA, ΦB, and ΦC. These signals provide edges at two timing points equally spaced over a 1 microsecond period. Phases ΦA and ΦB are nearly identical, except that ΦA starts and finishes at a low level, while ΦB starts and finishes at a high level.

Rising transitions of ΦA increment the OUTGOING BIT COUNTER. The WRITE DATA SHIFT REGISTER shifts or loads on falling edges of ΦB, insuring that the shift/load level input of the register does not change simultaneously with its clock input. The least significant two bits of the OUTGOING BIT COUNTER are used in the following sequence to generate clock and data pulses with the

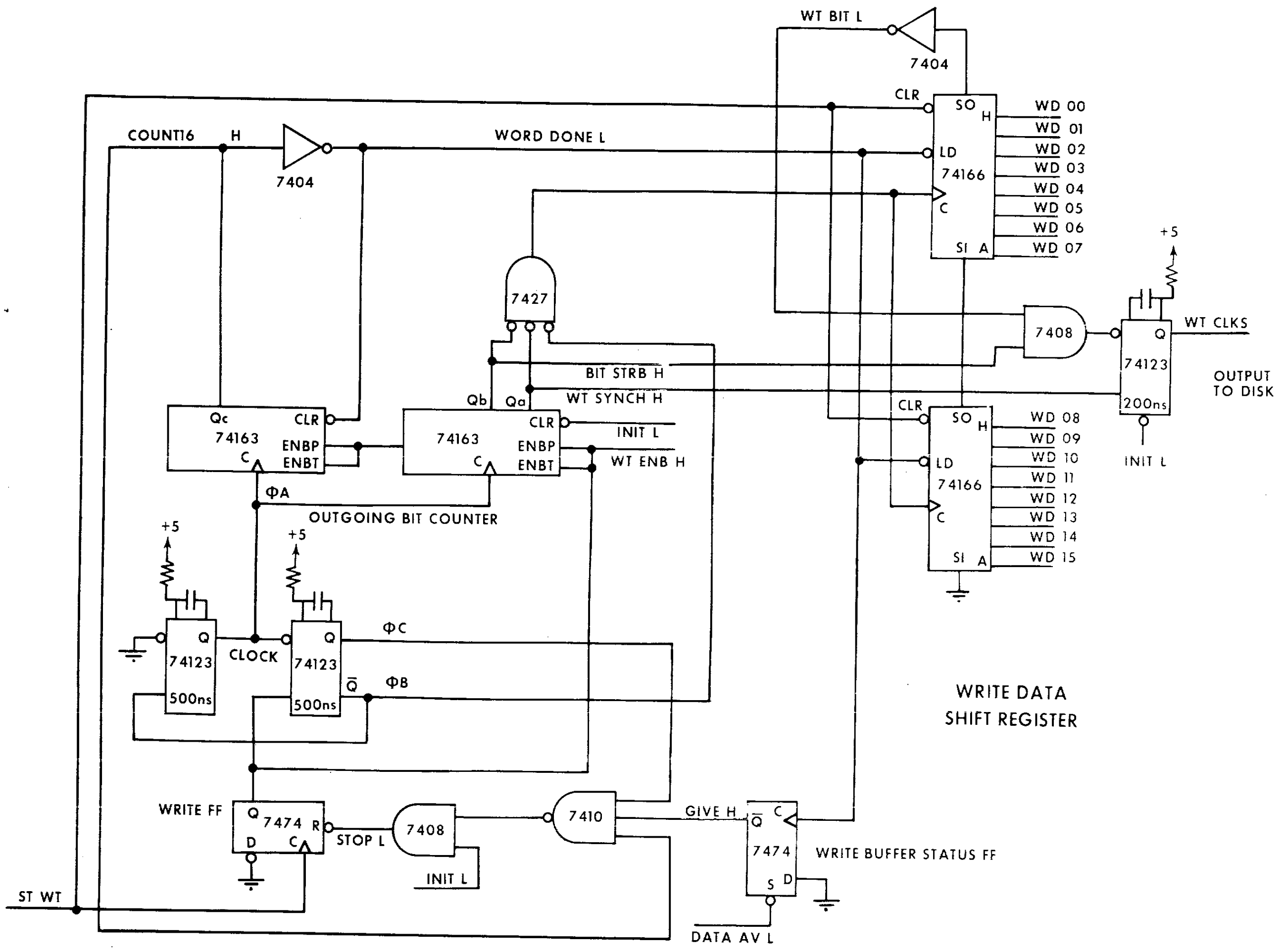


Figure 6—The Write Module Circuit (Write Buffer not shown)

proper timing:

States		Action
QB	QA	
0	0	Advance or load shift register.
0	1	Generate synch pulse on drive's WT CLK line.
1	0	Enable data to WT CLK one-shot.
1	1	Generate a pulse on WT CLK line if shift register data bit is one.

Finally, COUNT16 H asserts the clear input of the OUTGOING BIT COUNTER and the load input of the shift register. The next falling edge of ΦB then loads the shift register and the subsequent rising edge of ΦA resets the counter.

The load operation transfers the contents of the write buffer to the shift register and asserts GIVE H. In order to insure an uninterrupted flow of data to the disk, the write buffer must be reloaded before the next assertion of COUNT16 H and ΦC H. If DATA AVL L does not reset the WRITE BUFFER STATUS flip flop before COUNT16 H is asserted again, STOP L is asserted, WT ENB H falls, and the CLOCK stops after one final rising edge on ΦA (which clears the counter).

SECTOR AND TRACK CONTROL

The Memorex 651 disk platters can be formatted for 32 sectors per track, but the information capacity is reduced to 2.2 megabits, or 88 percent of the maximum capacity, when this is done. With the disk divided into only 8 sectors, 97 percent of maximum capacity is achieved. Since every sector must be written to completion, division into 8 sectors

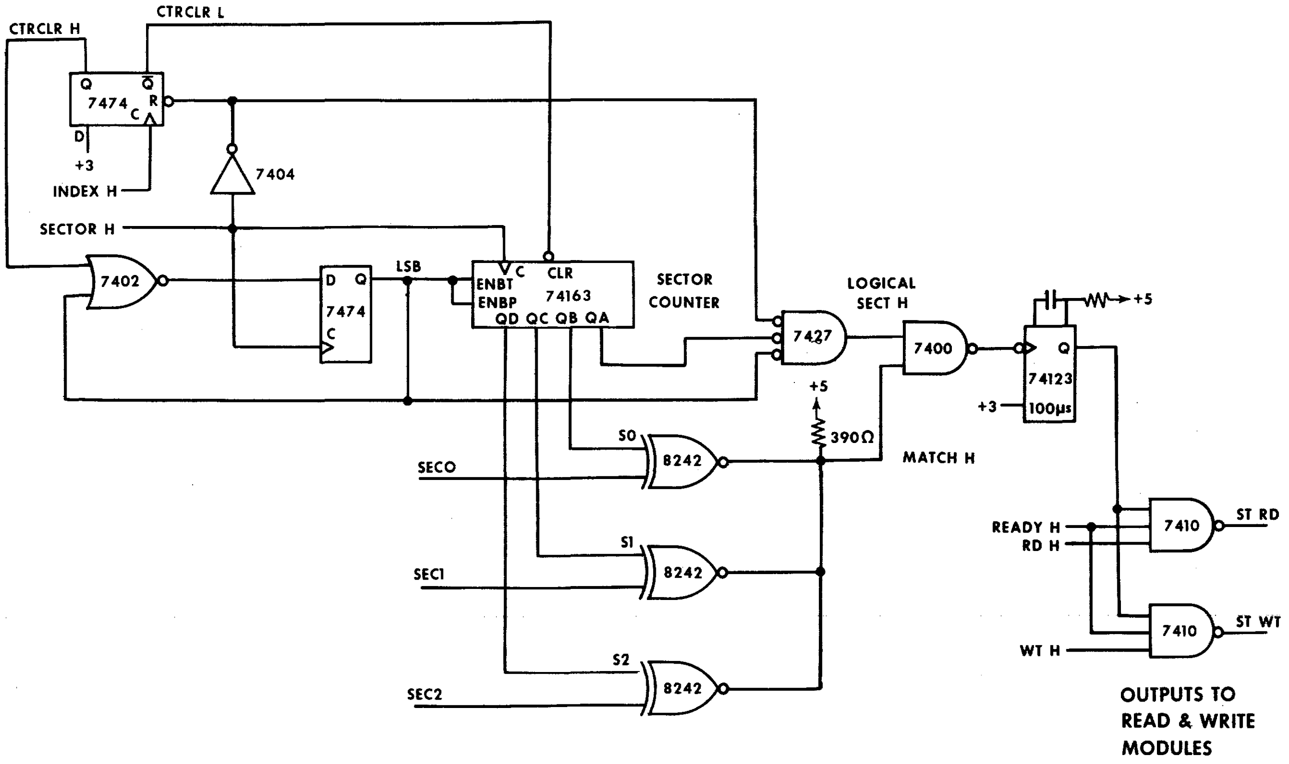


Figure 7—The Sector Match Module

offers a reasonable compromise. The Sector Match Module divides the disk into 8 logical sectors.

The SECTOR COUNTER of Figure 7 is a five bit counter made from a four-bit synchronous counter and a single flip flop for the least significant bit. The three most significant bits define the logical sectors. The index pulse asserts CTR-CLR so that the next sector pulse will reset the SECTOR COUNTER to zero. Whenever a sector match occurs with the sector (SEC0, SEC1, SEC2) specified by the current operation, a pulse is generated for ST RD or ST WT if either is enabled.

Track addressing is handled by the Track Address Module shown in Figure 8. Once the ACTUAL TRACK POSITION counter is set to the correct value, the computer need only specify the target track address (T0,T1,T2,T3,T4,T5) and pulse NEW TRK H. The Track Address Module compares the target track and the current track, and, after waiting 200 ns to complete the comparison gates, a 10 microsecond pulse to either STEP IN L or STEP OUT L. At the same time, a 10 ms delay is generated to allow the head settle. While the result of the previous comparison was not "equal," the falling edge of the settling pulse is gated to the up or down count input of the ACTUAL TRACK POSITION counter by MVD L, continuing the motion.

The program should initialize the ACTUAL TRACK POSITION counter before the first disk access by asserting TINIT H and NEW TRK H. TINIT H sets the target track (T0, . . . , T5) to zero and starts the step out sequence cycling the head toward the home position. A microswitch in the drive senses when the head is home, asserting TRACK0 H which clears the ACTUAL TRACK POSITION counter.

The HD LD H signal must be asserted for the duration of any read or write activity. A pulse is generated on the rising edge of HD LD H which provides sufficient delay for the pressure pad to engage the platter.

When any head motion is taking place, the Track Address Module asserts MOVING L to prevent the controller from attempting to read or write the disk until the head has settled. INDEX H triggers a 160 ms pulse from a retriggerable monostable multivibrator. As long as the drive is up to speed, this pulse is continuously retriggered, enabling READY H, which may be read as a status bit by the computer. Finally, when the unit has READY H asserted and neither RD ENB H or WT ENB H is asserted, DONE H is asserted.

STATUS SIGNALS

If this interface is used in a program transfer controller, GIVE H and TAKE H signal the computer to read or write the device's buffer. In a DMA controller, these signals instruct the DMA channel to transfer data to or from memory.

Read and Write are not symmetric with respect to the action taken when a buffer has not been transferred in time. The Write module has its own clock, enabling it to detect the end of a write cycle and shut itself off. Read, however,

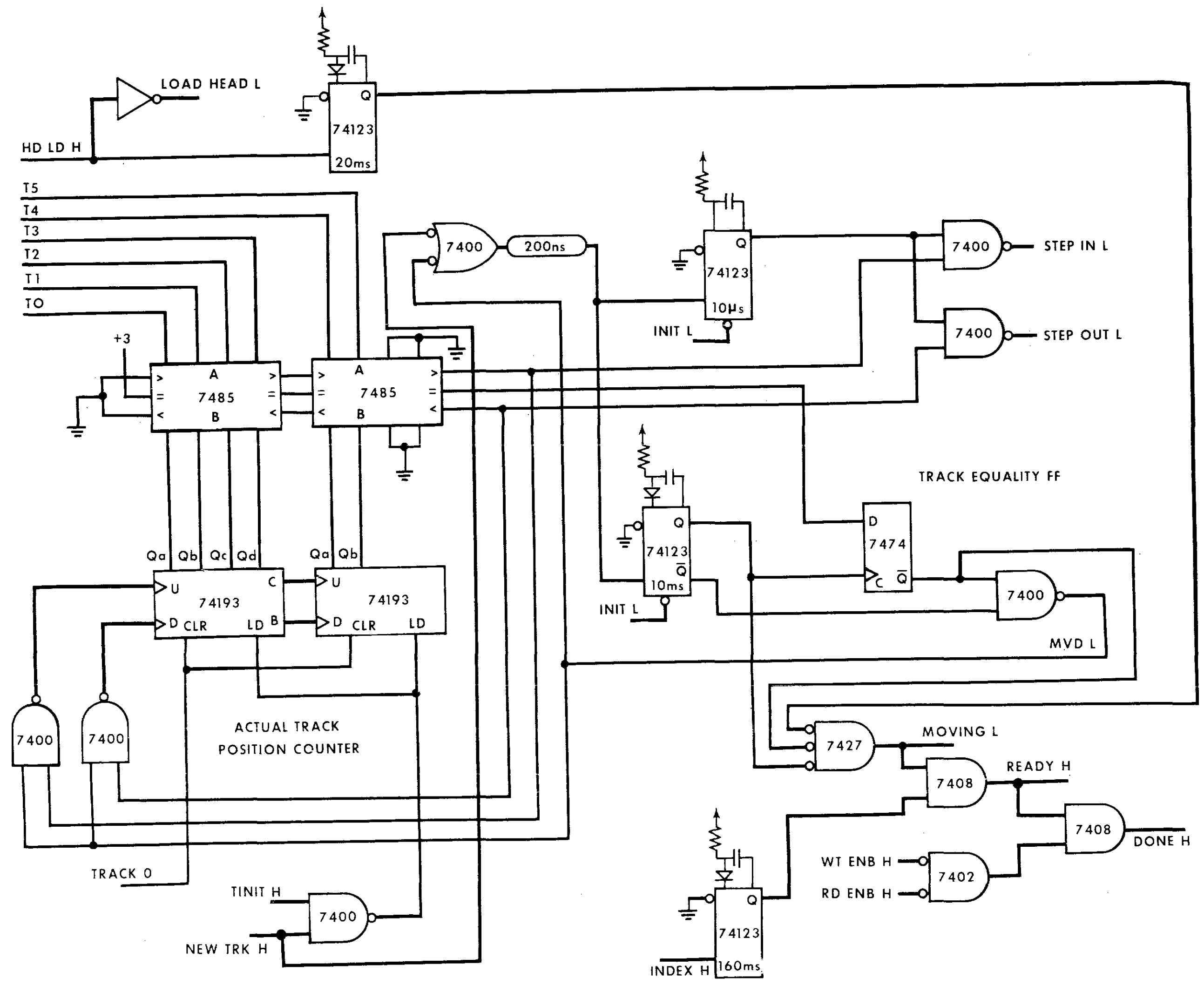


Figure 8—The Track Position Module (Head Load timing included)

is clocked by the data, and so may receive no further pulses after the last valid data has been read. Therefore the computer or DMA channel, which knows when the last datum has been read, must reset the READ flip flop by deasserting the enabling input, RD H. DONE H signals the completion of a disk operation, i.e. Had Load, Move Head, Read, or Write, and will normally cause an interrupt to the host computer.

THE PROGRAMMER'S VIEW

The four modules described here are the basis of any design for an interface to a specific computer. The details of how specific control lines are driven, or how transfers of data are accomplished, may be found in the computer manufacturer's interface manuals.[6,7]

We have built a program transfer controller for the Memorex 651 and the Digital Equipment Corporation PDP-11 using these modules. (Figure 9) The interface plugs directly into a PDP-11 Small Peripheral Slot and requires only the DEC M782 interrupt and M105 bus address modules.

The PDP-11 interface has three program-addressable registers:

(1)	CSR	Control and Status Register
(2)	TAR	Track Address Register
(3)	DBR	Data Buffer Register

Through these registers, the computer program can initiate activities on the disk, transfer data, and determine the status of the disk. In addition, the interface can generate two different interrupts to the computer: a "status" interrupt

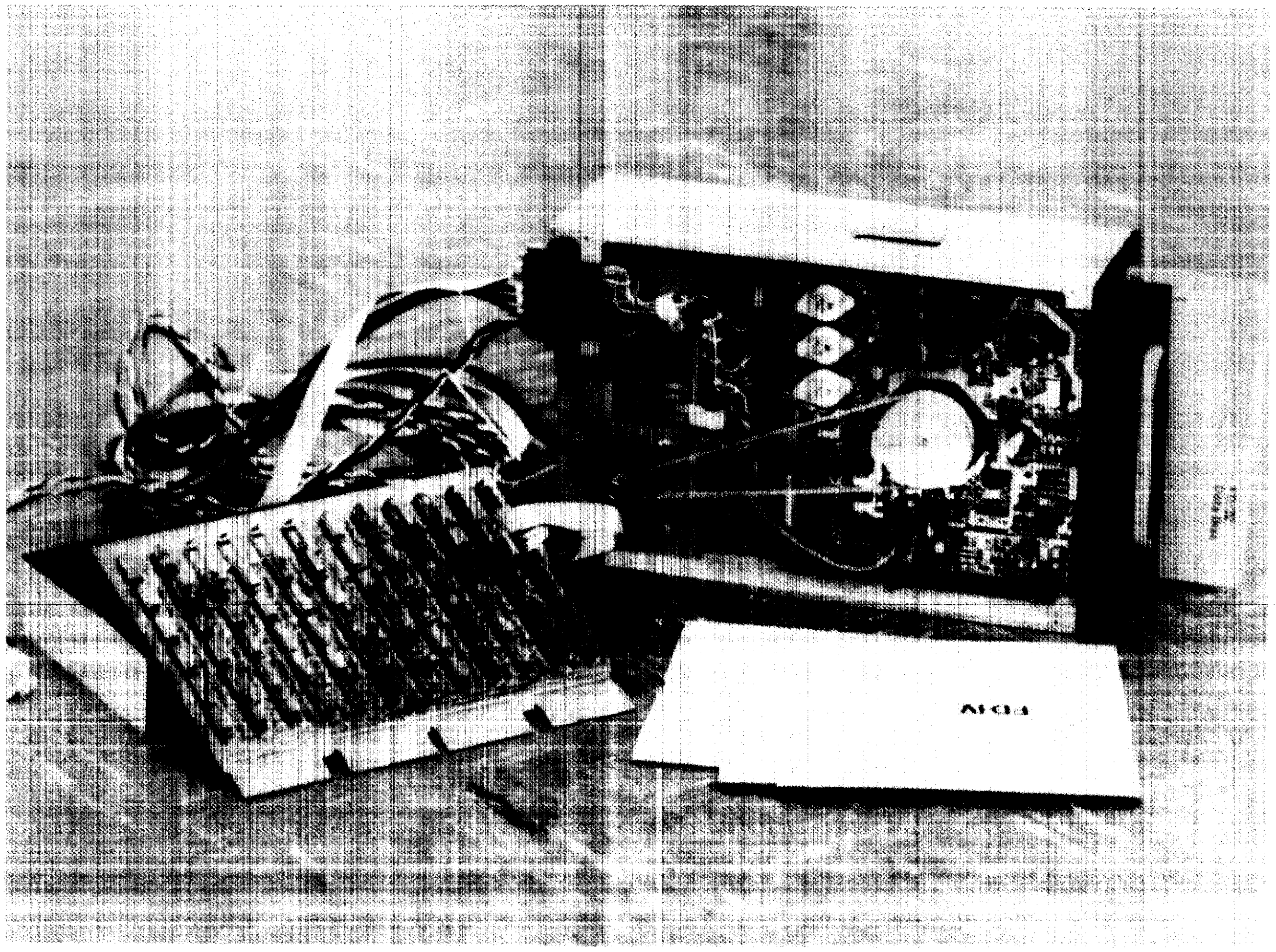


Figure 9—Interface collage board together with a view of the drive electronics and some sample platters

which indicates the completion of a command, and a "data" interrupt which is a request to transfer the next word of data to or from the DBR.

The commands to the interface are to read or write sector number (SEC0, SEC1, SEC2), to move the head to track number (T0, . . . , T5), and to load or unload the head. The status information available to the computer is head loaded or unloaded, present disk sector, actual track address, ready for next command, require next data transfer, and head home.

SUMMARY

We have presented a design for a simple disk controller for a Floppy Disk. The low cost of both platter and drive make the Floppy Disk an excellent substitute for a conventional disk on a small minicomputer system. The device can be used for both data and the resident operating system since in our experience media wear has proven low and data reliability relatively high (approaching the quality claimed by the manufacturer).

ACKNOWLEDGMENTS

The authors wish to thank C. Minter, R. Tuttle, J. Zornig, M. C. van Leunen and C. Carr of Yale University for their assistance. We would also like to thank P. Weiner of Rand Corporation for his contributions to the success of the project.

REFERENCES

1. Davis, Sidney, "Disk Storage for Minicomputer Applications," *Computer Design*, Vol. 12, #6, June 1973, pp. 55-66.
2. Painke, Helmut, "Initialization of Microprogrammed Machines," *Proceedings of the 5th Annual Workshop on Microprogramming*, Sept. 25, 1972, pp. 3-7.
3. *Interface Manual for the Memorex 651 Floppy Disk Drive*, Memorex Corporation, Santa Clara, California.
4. Hawkins, Joseph, *Circuit Design of Digital Computers*, Wiley, 1968, Chapter 9 (magnetic Surface Recording).
5. *The TTL Data Book*, Texas Instruments Incorporated, P.O. Box 5012, M.S. 84, Dallas, Texas.
6. *The Peripherals Handbook—PDP11*, Digital Equipment Corporation, Maynard, Mass. 01754.
7. Peatman, J. B., *The Design of Digital Systems*, McGraw Hill, 1972, Chapter 6.

Planning and design of data communications networks

by WUSHOW CHOU

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

The worldwide trend in computers in the 1970's has been the movement from off-line to on-line systems and the growing integration of computing and communications. It is expected that 70 percent of all computers will be connected to terminals via communication facilities by 1980. This trend has stimulated new applications for integrated systems, generically termed data or computer communications systems, and has broadened the current user base for existing applications.

As a result of this, communications systems are becoming increasingly larger and complex. In addition, innovations in system design concepts, hardware features and transmission services have accelerated. It is not easy for a communications network designer to cope with the complexity and the innovations that are being continually introduced. When he is just becoming accustomed to character oriented data communication control procedures as a way of life, he is told that the advanced control procedures are bit-oriented. When he is just about to understand the concept of store-and-forward packet-switching for computer communications he is exposed to "ring" structured network concepts. When he has begun feeling comfortable with multiplexers, he is faced with programmable concentrators and front-end processors with endless options. When he is just about to think that modems and analog signals are natural ways to transmit digital information, he realizes there will be digital transmission services and digital interface units. When he is still struggling to understand and distinguish among the various terrestrial transmission services offered by common carriers, he is confronted by the emergence of domestic satellite communication services that may save him money. The list of these dilemmas is endless. A system composed of several hundred terminals used to be classified as "large"; now there are systems with thousands of terminals. Many existing systems are experiencing a rapid growth in traffic volume, and a corresponding increase in response time. It is fair to say that almost every data communications analyst and network designer has his share of worrying and uncertainty about his system. Rules of thumb obtained from previous practical experience alone are no longer adequate except for very simple cases. Or, for some situations, there is no such rule at all.

How should a manager conduct the planning or upgrading of a modern data communication network? How can a designer or an analyst evaluate the best system concept and design strategy for his system? How can a user select the best cost/performance communication devices and transmission services for his system? What are the impacts of new technologies and the proliferation of new devices, new services, new tariffs, and new vendors? Are there design and analysis tools available to assist the designer or analyst make his decision?

The purpose of this paper is to present concise answers to the questions posed. These answers are an overview of the state-of-the-art techniques and approaches in the planning and design of data communications networks. Rules of thumb and simple concepts that are readily available in trade magazines are not emphasized. Instead, we will concentrate on the new, sophisticated techniques which have been specially developed to handle the complex problems present in modern data communications networks, and are used by leading experts in the field.

Some of the above mentioned questions are treated in detail in Lynn Hopewell's, "Planning in the Data Communications Environment,"[1] Pat McGregor's, "Effective Use of Data Communications Hardware,"[2] Mario Gerla's, "New Line Tariffs and Their Impact on Network Design,"[3] and Aaron Kershenbaum's, "Tools for Planning and Designing Data Communications Networks."[4] Another purpose of this paper is to provide linkage and continuity among these four papers. Terminology or concepts that are mentioned in this paper but not discussed in detail in this paper or any of the above four papers is likely to be found in James Martin's, *System Analysis for Data Transmission.*[5]

MANAGEMENT AND PLANNING

In most large communications networks currently in operation, costs can easily be reduced by 15 percent or more with only minor alterations in the network. The cost savings can often reach 30 percent or more if reoptimization of the whole network is allowed. Furthermore, the cost reductions can be realized without degradation in performance. Indeed, both cost and performance can often be improved simultaneously.

The implication of the above statement is that the existing networks are not properly planned and designed. Why is this so? If one has to point a finger, the traditional corporate structure is to blame.

Traditionally, corporate communications systems have been left in the hands of middle or low level managers, mostly with limited technical background. With the emergence of data communications, which demands a broad range of knowledge in both practical engineering and advanced theory, the top management finds it even more undesirable to exercise direct control over the planning and operation of communications networks. Let us now examine the consequences of this lack of interest and understanding of the complexity of the problem.

Middle level technical managers who often have the major responsibility for planning and operation face many difficulties. First, these network managers are usually technically oriented. They often do not have enough training in management. They are also likely to have the characteristics common to almost all technical people, overestimating their ability and overcommitting themselves for technical projects.

Making the whole planning and operation process worse is the fact that their hands are usually tied. It may be relatively easy to get approval from the top management for tangible expenses such as buying communications equipment and paying line costs. It is considerably harder, however, to obtain funds for less tangible expenses, such as program development and seeking expert assistance during the planning and design process. Thus, they must make compromises within the limitations of their budgets. Furthermore, they do not want to risk user dissatisfaction. As a result, the tendency is to overdesign networks, using more lines and equipment than is actually necessary. So the small saving in the planning stage often results in a large increase in the cost of the final network. Also, the overdesign may not guarantee satisfactory performance.

Due to their relatively low level in the organization hierarchy, the network managers do not want to chance any possible blunder. This usually means that they would hesitate to initiate any innovation or change unless it is absolutely necessary.

Another difficulty is the lack of coordination among different groups providing technical assistance to one another in a broad data processing system or a larger corporate communications system, of which the data communications network is a part.

Probably the single most significant factor that contributes to the inefficient designs is a general underestimation of the complexity of data communications. Data communications networking is an evolving, new technical field. To be able to master this field, one needs to know computer software and hardware, communications hardware, transmission facilities, line tariffs, human psychology, queueing theory, statistics theory, communications theory, advanced computational techniques, and the most advanced network optimization techniques. Some of the advanced techniques may be the result of the most current research. He must be aware of the new developments in the field. In addition, he must be clever and think innovatively. There are not many people who possess all these capabilities. Without knowing this situation, a big corporation's management is likely to think that somebody in the company should be able to develop an expertise in the area of data communications networks, or at least it should be easy to hire someone with this expertise. In some cases, usually out of desperation, assistance from consultants is sought. However, with the notion that any of the many data communications consultants can do the job, technical competence does not always play enough of a role in the selection of a consultant. A consequence of this is that one may not get the necessary expertise.

This whole section has been discussing the negative part of the management planning process. What is the positive side? In this author's viewpoint, but not the author's alone, the improvement can be achieved by upgrading communications management to the corporate management level. The network manager with management background should force himself to be technical; the network manager with technical background should force himself to acquire management ability. There is actually a trend, which is taking hold, though slowly, toward the upgrading of the communications management and the merging of both technical and managerial talents into the management. One can observe this trend by the fact that the chief executive officers of some of the new specialized common carriers have reputations of superb ability in both the technical field and management.

REQUIREMENT ANALYSIS AND DESIGN CONSTRAINTS

Successful implementation of a data communications system depends to a great extent on the thoroughness of the data traffic analysis and the user requirement analysis.

Gathering traffic information is the most tedious part of the planning. Traffic information should be collected from every user by means of current measurements and future projections. To be included in the traffic information for each terminal and for each type of message or transaction are distributions of the number of transactions per unit time during the peak hour, average day and peak day, input message length distribution (number of characters per message), output message length distribution, and priority. It is almost impossible to accurately measure and project this information. However, one should try his best. It is often helpful if the planner visits the users to assist them and to validate the information supplied by them.

Even more uncertain than the traffic statistics and projections are the users' requirements on network performance. In general, users do not know exactly what they want. Sometimes they may demand a level performance which is practically impossible. Other times, they demand a performance level that they may not need but must pay a high price to attain. If cost is not a concern, any user would like to see negligible response time and almost perfect network reliability. Apparently, no one can afford to pay for such level of performance. But users usually do not have the feeling for the cost/performance relationship. It is the planner's and the designer's responsibility to educate the users, to show them the relationship between cost and performance, and thereby assist them in modifying their performance requirements to reasonably obtainable levels. The traffic information and the users' performance requirements thus form the constraints for the design.

PERFORMANCE CRITERIA AND CONSIDERATIONS

A general goal in designing a data communications network is "to design a minimum cost network satisfying performance requirements or criteria." A common slogan in this field is "improving the cost/performance ratio." What is performance? It means different things to different people. For a well designed system, the performance should be measured by the following criteria: blocking probability and/or message response time, traffic capacity or throughput, network reliability, transmission error rate, and sensitivity to variations in traffic level. Knowing the traffic bottleneck is also an important piece of information, especially for future expansion and upgrading. However, it is not generally used to measure current network performance.

Blocking probability

This criterion is used to measure the promptness with which a data communications system responds to calls from dial-up terminals. It can be defined by any of the following three factors:

1. At least "A" percent of calls obtain access to a computer port within "B" minutes (this definition is used for systems in which to have a connection, the terminal must first dial to a switchboard; an operator then attempts to connect it to an empty computer port).
2. At least "C" percent of calls obtain access to an empty port on the first attempt.
3. "D" percent of calls obtain access to an empty port with no more than two attempts, three attempts, etc.

The parameters A, B, C, and D are constants determined by user requirements or network planners. For example, "A" can be 99, "B" 5, "C" 95, and "D" 99.

Message response time

This criterion is used to measure the promptness with which a system responds to terminals connected to the system by leased or private lines. Message response times have different definitions at different parts of a data communication system. So far as the users are concerned, "terminal response time" and "overall response time" are most meaningful. The terminal response time is defined to be the time required from the instant the "transmit" or equivalent key on a terminal keyboard is depressed to the moment the reply message begins to appear at the terminal. This is the most commonly used criterion. However, it has serious drawbacks when used as a measure of the "service promptness," since a user may have waited a long time at the terminal before his message is keyed in. The overall response time is the elapsed time from the instant that a user or a message arrives at a terminal to the moment the user is completely served or the reply to that message is received. The response time requirement is usually defined as "average response time should be no more than X seconds" and/or "response time for at least Y percent of transactions should be no more than Z seconds."

System capacity or throughput

Capacity in its most liberal interpretation is often taken to mean the maximum amount of traffic, in terms of transactions per second, characters per second, etc., that a system can carry. Unfortunately, such a definition would be totally unrealistic since a user, attempting to send messages into a system operating at this capacity, would experience intolerably long delays from the time he inputs his message to the time he receives a reply.

For a more practical definition, the capacity is defined as the maximum traffic that a system can carry, while satisfying the blocking probability criteria and/or response time requirements.

Network reliability

While the failure rates, MTTF, and MTTR of the equipment and lines are often beyond the control of network planners, the network's reliability can usually be strengthened with proper network structures. Again, the definition of reliability varies according to usage. For most purposes, at least one of the following definitions is applicable.

1. Percentage of time a terminal can communicate with the central computer.
2. Percentage of time an office can communicate with the central computer (the office may have more than one terminal).
3. Percentage of time a terminal can communicate with any other terminal (this definition applies when there is direct inter-terminal communication).
4. Percentage of time an office can communicate with any other office.
5. Average number of terminals or offices that are connected to the network.
6. Average number of equipment failures, or the average man hours required for repair, in a day or other unit of time (this definition is usable for equipment maintenance crews).

Sensitivity

A system may behave properly if the traffic volume is within the projected range, but break down entirely if the traffic exceeds the volume for which the system was designed. Thus, a good planner should be concerned with the effects that the system would experience if the actual traffic is above the projection. He should make sure that a small variation in the projection does not create intolerable response times, or blocking probability. He should create a curve like the one shown in Figure 5 during the planning process.

Transmission error rate

The transmission error rate is a function of message size, line conditioning, and hardware characteristics. It is more critical in a centralized data communications environment than in that of a distributed computer network. That is because the former usually does not have error detection or correction features, while the latter does.

As an example, suppose the error rate is not allowed to be more than one in every thousand characters. The designer or planner should calculate the error rate for his candidate system or network. If it is less than 0.001, he is safe. If not, he has two choices. One is to add error detection and correction features to the terminals or terminal controllers. The other is to redesign the system, i.e., use lower speed modems, and fewer terminals per multidrop line, etc.

Traffic bottleneck

After a system has been designed to satisfy specified traffic requirements, the traffic bottleneck of the system initially is not a characteristic of interest. However, if in the future the network has to be upgraded to handle more traffic, knowing where the traffic bottleneck is in the network helps to estimate the incremental cost of expanding the network's traffic handling capacity. For example, if the bottle neck is the high speed line between a concentrator and the central computer, it is quite simple to upgrade the network capacity, either by adding an additional line or by using modems of higher speed. On the other hand, if the bottleneck is at the central computer, the upgrading would be costly.

SELECTION OF COMMUNICATION DEVICES

With the advancement of solid state electronics, communications devices become ever more versatile and generous with options. There are numerous possible combinations available for improving performance, reducing communications cost and satisfying special requirements. However, these goals are not easily achievable. One must know what a vendor has "not" said, what devices are most effective for specific network structures or performance requirement, and how many of each particular device should be used.

A high speed modem can increase throughput, improve response time, and may even reduce costs, but one has to be aware that the transmission error rate is much higher on a line with a higher speed modem. Sometimes, this high error rate prohibits the use of high speed modems with non-intelligent terminals. A modem sharing unit can save cost in a system in which several terminals may be co-located in a same office and share the same multidrop line, but then network reliability will be lowered. On the other hand, a combination of modem sharing units and port or line sharing units may improve reliability as well as reduce costs. A multiplexer or a concentrator may reduce overall communications cost, but only under particular traffic and operational environments. Without proper planning, the introduction of multiplexers and/or concentrators will degrade network performance, and even increase the costs. For more details, refer to Reference 2.

SELECTION OF TRANSMISSION FACILITIES

All data communications network need transmission lines. The planner must choose between dial-up and dedicated lines, choose the right line speed, and calculate and compare line costs.

In the dial-up case, a terminal needs to be connected to the network by dial-up only when there is a need for actual communication. In the dedicated line case, a terminal is connected to the network via a leased or private line and is connected regardless of whether or not there is any actual transmission taking place at a given time. Dial-up connections are used if terminal locations are not fixed (like traveling salesmen's portable terminals), terminals do not belong to a same organization (like in a time sharing environment), terminals are used in remote batch environment with low utilization, or terminals are sparsely located with low utilization. Presently, the dial-up arrangement is either by direct distance dialing (DDD), foreign exchange, or WATS lines. (For the definitions and applications of these three, see References 5 and 6.)

The choice of terminal speed depends mainly on the application, performance requirements, available line tariffs, and cost. It may be determined by rules of thumb, experience, or standard speeds of terminals for the specific application. The simulation program described later and in Reference 4, can be used to determine response time given specific line speeds to see whether a specific speed satisfies a given requirement, and to evaluate cost/performance tradeoffs using different line speeds. In the same network, different terminals and communication devices may be connected to lines with different speeds.

Before the proliferation of specialized common carriers, such as MCI, it was relatively easy to select and interface a specific line type and to calculate line costs, since the user had the convenience of interfacing and dialing with a well defined transmission facility and common carrier. Now, with existing and forthcoming special common carriers of various types, numerous new line tariffs and modifications to old tariffs (AT&T's Hi/Lo Density, Digital Data Service, etc.), and the addition of domestic satellite communications service and digital transmission service, users are confused. (Communications consultants are happy, however.) The determination of line costs and optimal network topologies is harder than before. The least line cost for routing a line connecting two terminals or devices in a network is no longer necessarily a direct connection, or the one of minimum distance. In some situations, hand calculation is impossible and a special computer program is necessary. For more detail, see Reference 3.

NETWORK STRUCTURES

Network costs and performance depend greatly on the structure the planner chooses. In general, there is no easy way to determine a best structure. Repetitive simulation and design are required to determine the most desirable one. The following are the most commonly used network structures:

1. Point to point connection via dial-up.
2. Point to point connection with leased lines.
3. Multipoint tree-structured connection. The tree structure usually terminates at a multiplexer, concentrator, message switching processor or central computer. In general, the tree network and terminals are so connected that when one device is transmitting a message, it is transmitted to every other device connected in the tree. (This is in contrast with the ring structured network.) However, only the ones with proper identity would actually receive the message.
4. Multipoint ring structured connection. In this structure, terminals or devices form a ring or loop as shown in Figure 2. When terminal A is sending a bit to terminal B, no other terminal knows anything about this bit. (This is in contrast with the tree network.) If this bit information is not for B, B then passes it to C, and so on. At least one of the devices on the ring is a computer which is either the central computer or is a message switching computer capable of switching a message from one ring to another.
5. Multiplexed structure. Several low speed point to point lines or multipoint tree structured lines are terminated at a multiplexer; the multiplexer is then connected to a distant multiplexer with a high speed line. By this arrangement, every low speed line is connected to the distant site as though each had its own line, rather than sharing with other lines.
6. Hierarchical ring structure. Lower level rings are connected to higher level rings via a message switching computer.
7. Hierarchical structure (without rings). Terminals are connected to concentrators or message switching computers first. These computers are then either connected directly to a central computer or connected among themselves.

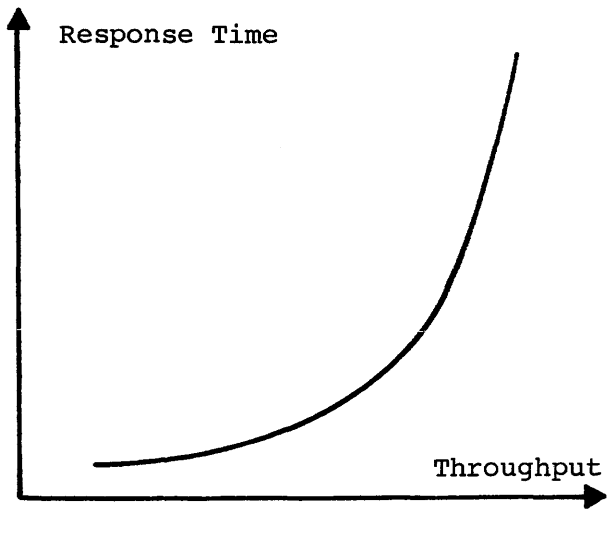


Figure 1—Response time vs throughput (Useful for performance sensitivity evaluation)

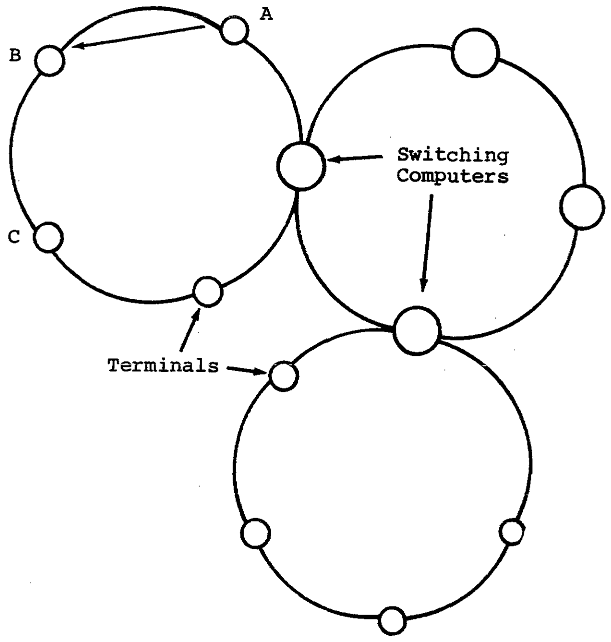


Figure 2—Ring-Structured network

DESIGN TOOLS

Due to the complexity of many data communications networks, hand analysis and design becomes almost impossible. Computer programs for various analysis and design functions are essential. The usefulness of such programs relies heavily on how convenient it is to repetitively run the programs. Thus, efficiency in size and running time is as important as accuracy. The following is a list of important

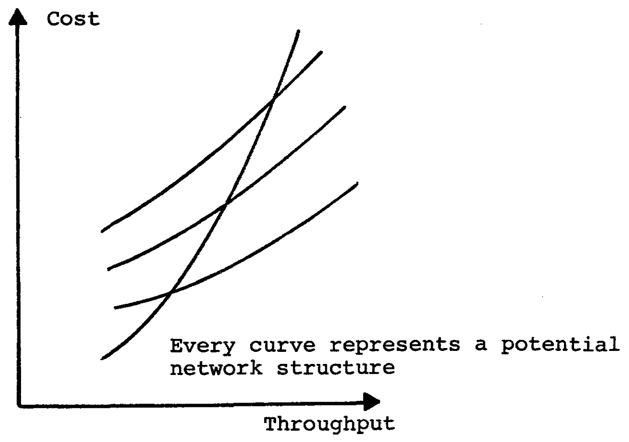


Figure 3—Cost vs throughput for a fixed response time (Useful for evaluation of network structure alternatives)

design tools:

Contention simulation program: Determining blocking probability

This program is useful for dial-up terminals. The output of the program should give the distribution of blocking probabilities for each of the following: number of users (terminals), number of ports, distribution of number of calls per unit time at each terminal, call holding time distribution at each terminal, and distribution of time intervals between two consecutive attempts to obtain a channel for the same call.

Central processor system configuration program: Verifying performance for specified CPU configurations

For specified CPU and peripheral device types, the program should indicate whether the specified configuration satisfies throughput and/or response time requirements. If so, the total time span spent by a message in the CPU and peripheral devices will be given. If not, the bottleneck causing the oversaturation will be indicated. Analytic queueing models, as opposed to brute force simulation and analytic closed form formulae, should be used to develop this module. Brute force simulation of the CPU is too complicated and too time consuming, with respect to both development and execution.

Network simulation program: Simulation of the whole data communications system

Given a network configuration and traffic requirements, the module should be capable of supplying terminal response time statistics to provide a response time—throughput relationship as a function of network configuration. This is done by simulating a whole system, including regular and intelligent terminals, multidrop lines, concentrators, trunk lines, DCP's and CPU's. For effectiveness and efficiency, simulation, analytic formulae, queueing models, and empirical distributions should be judicially mixed into the program. Specifically, intelligent terminals, concentrators, and CPU's are to be simulated by queueing models or are to be described by empirical distributions. Lines and terminals are to be simulated.

Network design program: Concentrator and multiplexer allocation, terminal clustering, multidrop line topological design and economical analysis

Given the locations of terminals, concentrators, and CPU's; their basic characteristics; the traffic characteristics; and line utilization requirements; multiplexer and concentrator locations are selected; terminals are connected to the proper concentrator or CPU via a multidrop line in a cost effective manner. Thus, an important design goal is to implement optional heuristic algorithms into the program so that the program size and running time can be proportional to the number of terminals. (In general, the size and running time grow quadratically or cubically with the number of terminals.)

Network reliability/availability program: Calculation of network reliability

Given element failure rates (or MTTF's and MTTR's), the module calculates network reliability criteria. A combination of simulation and analytic techniques should be used to ensure the effective determination of reliability for networks with thousands of terminals within reasonable computer time. For more details on design tools, readers are referred to Reference 4.

DESIGN STRATEGIES AND COST/PERFORMANCE TRADEOFFS

It is the network planner's responsibility to assist users in defining their requirements and to design a least cost network while satisfying the requirements. To do a good job for a large network, one needs to develop a set of curves to weigh and compare the tradeoffs for cost/performance and for design alternatives. Design tools described in the previous section are extremely useful for this purpose.

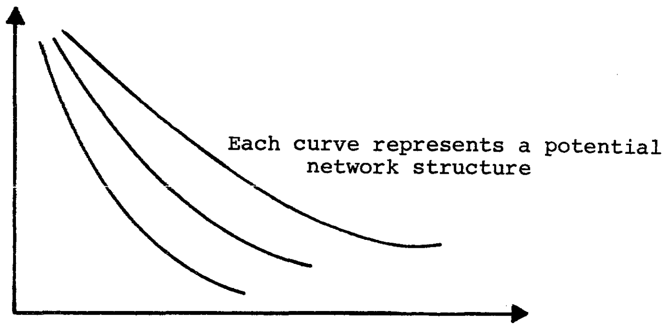


Figure 4—Cost vs response time for a given throughput requirement (Useful for evaluation of network structure alternatives)

Evaluation of design alternatives

By choosing some of the network structures given earlier and by using different line speeds, one can develop several sets of curves like those shown in Figure 3 and Figure 4. For a specified response time requirement, each curve in Figure 3 represents the cost-throughput relation for a specific network structure. For a specified network throughput, each curve in Figure 4 shows the cost-response time relationship for a specific network structure. From these curves, one can determine the most cost-effective network structure.

Evaluation of cost-throughput tradeoffs

Figure 5 shows the relationship between cost and throughput for a fixed response time requirement. With these curves, the network designer can help users to decide how much they are willing to, or must, pay for throughput in the network.

Evaluation of cost-response time tradeoffs (CPU response time, network response time, or CPU and network response time)

In Figure 6, each curve represents the cost-response relationship for each specified throughput requirement. These curves help the user to determine how much he is willing to pay for the response time that he will get, and help the designer to determine the best combination of CPU configuration and line configuration.

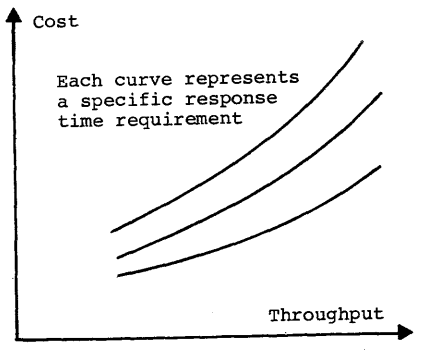


Figure 5—Cost vs throughput with response time as a parameter (Useful for evaluation of cost-throughput tradeoffs)

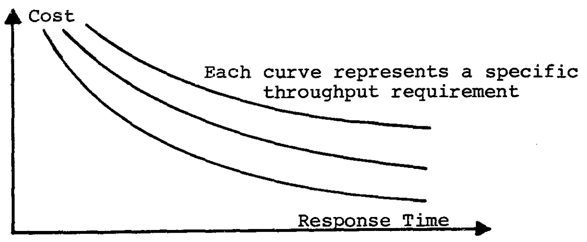


Figure 6—Cost vs response time with throughput as a parameter (Useful for evaluation of cost-response time tradeoffs)

Evaluation of cost-reliability tradeoffs

Reliability is another term that users do not quite know how to define. There may be many schemes for improving network reliability. To evaluate them, one must develop a curve to show the incremental cost one must pay for improved network reliability.

Derivation of throughput-response time relationship

Since traffic estimates are rarely accurate and future growth is even harder to predict, network planners need to know how sensitive network performance is to traffic variations from projected levels. Figure 1 shows the relationship between performance and throughput, with performance measured in terms of response time.

REFERENCES

1. Hopewell, L., "Planning in the Data Communications Environment," *Proc. of 1974 NCC&E.*
2. McGregor, P., "Effective Use of Data Communications Hardware," *Proc. of 1974 NCC&E.*
3. Gerla, M., "Line Tariffs and Their Impact on Network Design," *Proc. of 1974 NCC&E.*
4. Kershenbaum, A., "Tools for Planning and Designing Data Communications Networks," *Proc. of 1974 NCC&E.*
5. Martin, J., *Systems Analysis for Data Transmission,* Prentice Hall, 1972.
6. *Datapro 70,* Datapro Research Corporation, 1973.

Management planning in the data communications environment*

by LYNN HOPEWELL

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

A dominant trend in the computer business is the growth of data processing systems which are involved with data communications.[1] It is estimated that by 1980 70 percent of all computers will use communications. Many organizations are now using or planning the use of teleprocessing systems. For those with such systems in operation, the experience has been one of rapid expansion and growth as the organization learned to appreciate and use them.

This rapid growth has increased the complexities of planning both for teleprocessing systems in general and for the data communications subsystem in particular. Data communications is now the most dynamic activity of once placid telecommunications groups in large organizations, and indeed, is now often the object of reorganization efforts generated by the problems of such a rapid rate of change. Rapid growth, the relatively long lead time for implementing changes, and the large investment and operating costs of data communications have caused heavy emphasis to be placed on the advanced planning function. Any data communications support group that has felt the wrath of users when experiencing an unanticipated increase in response time of an inquiry system, for example, will be determined not to be caught flatfooted again.

This paper will present a conceptual framework for thinking about planning problems which I found useful in recent consulting assignments in data communications planning for large organizations. This conceptual framework has been tested in the last year on three major organizations: a large diversified financial conglomerate, a major manufacturer, and a high technology oriented government agency. In each case, the contribution of the conceptual framework has been to provide a way of thinking about planning problems that provides insight into their interrelationships. The framework is divided into a *data communications process*, a *planning process*, and *organizational considerations*. Some typical data communications planning problems encountered are also discussed.

THE DATA COMMUNICATIONS PROCESS

Figure 1 is conceptual view of the data communications *process*. Process simply means all those organizational activities that concern managing the data communications subsystems which are part of larger teleprocessing data processing systems. This includes both day-to-day activities concerned with operation of inplace systems, and all future oriented planning activities. The process primarily consists of the interaction of the following elements.

Tools

Tools are the individual technical skills, techniques and knowledge that are found in the data communications environment-software, telecommunications, simulation, statistics, etc. These tools are necessary no matter what the time orientation of any task is—short or long range.

The user

Every data communications support group has a user. The ultimate user may be, for example, the major lines-of-business divisions in a large multimarket company. Or, more often, the practical "user" may be a surrogate for the ultimate user; for example, the applications programming groups typically found in the line divisions. These data processing groups act as interpreters of requirements between the business user and the various technical system support staffs.

Other system processes

Data Communications is only a sub-system of a larger data processing system. Depending on the task and relevant time frame, the data communications process needs knowledge about, and must interact with, other systems processes. For example, system response time is affected by both the data communications sub-system and by the CPU sub-system.

The data communication process interacts with these elements with an intensity that is proportional primarily to the time frame relevant to the problem or task that is being "processed". Specifically:

Time Frame	Element
Short Term:	Tools
Medium Term:	Tools and User
Long Term:	Tools, User and Other Sub-systems

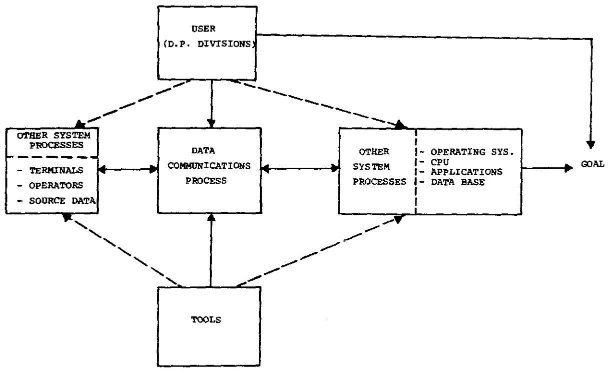


Figure 1—Data communications process

To see this more clearly, the data communications process must be viewed with respect to the planning process discussed below. A major thesis discussed later is that it is very sensible for the data communications staff to do their own short range planning, but that medium range planning requires interaction with user planning activities, and that long range planning only makes sense from a total systems standpoint.

THE PLANNING PROCESS

The planning process has been arbitrarily divided into short range, medium range, and long range. A case could be made for combining the latter two, but I believe there is a significant shift of emphasis. The three different planning processes can be usefully characterized by differences in data communications element interactions, goals, degree of uncertainty, constraints, and technology.

Short range

1. *Process Interaction.* The primary element involved here is the use of tools to solve very specific, limited tasks involving day-to-day operations.
2. *Goal.* The goal is narrow, simply to make the system work.
3. *Uncertainty.* Relatively little. No system or structural changes. Dealing with well defined parameters.
4. *Constraints.* Highly constrained; few alternatives. At most, a process of "fine tuning", eg: changing a few lines in the network, software maintenance, etc.
5. *Technology.* Technology well in hand; at most, unit substitutions possible, eg: a new modem.

Medium range

1. *Process Interaction.* Now problems are beginning to be anticipated, and more interaction with the user element is required. For example, traffic volumes are increasing. The user is the best judge of "how much" and must furnish these projections for use in a network reconfiguration.
2. *Goal.* Here the goal is to keep the system working under the stress of change. Changes confined to subsystem optimization and cost minimization, no major system capabilities altered.
3. *Uncertainty.* Since we are now looking ahead, solutions must be examined for sensitivity to changes in assumptions. What if traffic grows faster than anticipated? What will be the actual costs of a new device one year from now?
4. *Constraints.* More choices possible, perhaps some intersub-system optimization possible. For example, use of higher speed lines would result in cheaper networks which could be traded against possible new terminals. However, no major system changes likely.
5. *Technology.* Now feasible to examine technological change. For example, the use of concentrators in the network, or a communications front-end at the CPU.

Long range

1. *Process Interaction.* Now not only the user and tools are involved, but heavy interactions with all other subsystems is required. Consideration must be given to the tradeoffs and optimizations viewed from the end-to-end system as a whole.
2. *Goal.* Here the goal is not only maintenance and cost minimization, but can include benefits expected from major data processing architectural developments expected. View is data processing system oriented rather than data communications, operating system, etc.
3. *Uncertainty.* High. Planning consists of projecting numerous alternative scenarios. Each sub-system technology regards developments in every other subsystem as more uncertain than their own. Business intangibles must be considered.
4. *Constraints.* Few. Limited by value of service to user.
5. *Technology.* Limited only to the degree to which the state-of-the-art is pursued. System design must consider all meaningful developments in each subsystem, their interactions, and architectural consequences.

ORGANIZATIONAL CONSIDERATIONS

When considering the technical planning process, it would be insufficient to address only technological issues, for the process is really a highly interdependent relationship of technology and organizations. Weakness in either area can cause planning to be ineffective.

Technical planning can be thought of as the application of technological tools to problems *through* organizational design.

Technology
—Provides *tools* for analysis
—Provides *methodology* for applying tools
Organization
—Provides structure for applying tools and methodology to problems.

Indeed, organizational design can significantly influence whether problems are diagnosed or identified at all.

The broad question of how to organize the data processing function in organizations has been treated elsewhere.[2] The viewpoint taken here is that of a data processing technical support group organized as a service function serving line business operations. Such an arrangement is common in large multidivisional companies. Figure 2 illustrates this type of relationship. Within this support group are found the technical subsystem specialties necessary to support the broad data processing function, including the data communications specialty. Such an organizational approach enhances communications among the subsystem technicians, but creates barriers between these specialties and the user. These barriers are a major cause of difficulties in planning.

Thompson[3] has shown that most organizational designs manifest one characteristic in common. They establish special organizational units to deal with the major business environmental contingencies. More generally, organizations faced with heterogeneous task environments seek to identify homogeneous segments and establish structural units to deal with each. Thus, an effective method bridging the barriers created by one way of organizing the data processing support function is the creation of ad hoc methods of communication directly between technical subsystem specialists and the user. Special service representatives, committees, and lunch conversations are all used for this purpose. The surmounting of the barriers between support groups and the user is a primary necessity for effective medium and long range planning.

DISCUSSION

With the above framework for analysis in mind, some problems relating to planning that have been found to be common in numerous organizations are discussed below.

Short range: Tools

1. *System Models.* Amazing as it may seem, many organizations operating large, expensive, and complex teleprocessing systems and networks have no useful quantitative understanding of how their systems work and where bottlenecks will occur with rising transaction volumes. Often the lack of a model results from inappropriate attempts to build one. Advanced planning requires the ability to examine many assumptions, yet many complex teleprocessing system modeling efforts misuse simulation techniques, resulting in unwieldy and unusable models. An adroit blend of simulation and analysis usually yields better results.[4]
2. *Operating statistics.* The problem with most attempts to collect statistics on the operation of teleprocessing systems is that they generate an overabundance of irrelevant information. The idea seems to be that of creating an infinite pool of data into which a manager could dip and find any information he wants. It is more often a pool in which he drowns. Before any statistics are collected, an explanatory model of the decision process and the system involved in it must be constructed and tested.[5]

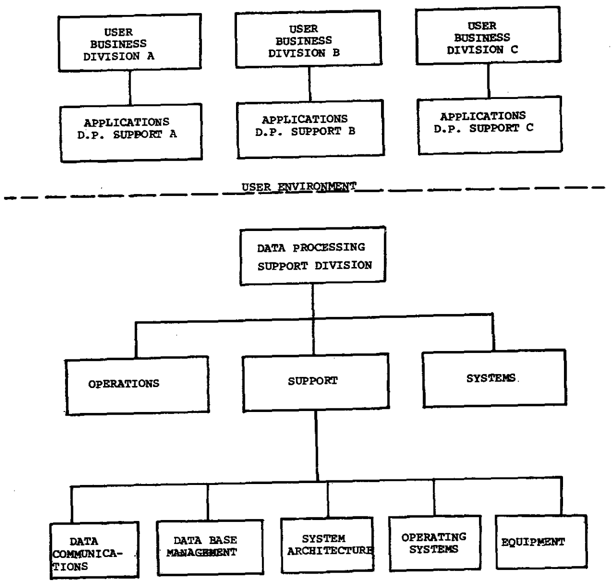


Figure 2—Typical data processing support organization

3. *Systems Engineering Methodology.* In spite of the profuse occurrence of the term "systems analysis" in the data processing environment, I have found that a fundamental barrier to success in planning has been the lack of appreciation of real systems engineering methodology.[6] The systematic consideration and analysis of total system alternatives is an underdeveloped skill in data processing, particularly in the non-engineering environments of banks, insurance companies, and the like. Sometimes this occurs because of the isolation of the teleprocessing subsystem technicians from each other that can occur in a large organization. Two instances have occurred recently in which the splitting of a massive 370-165 data base into two CPU's was averted at the last minute, after considerable effort had gone into the procurement process, by improving the response time of the data communications subsystem. This allowed easing of the CPU delay requirement and thus achieved more throughput with much less cost and complexity.

A related problem is the tendency to allow equipment vendors to do systems planning by default. Hardware vendors sell their own hardware, they are not oriented to, or capable of objectively identifying with the user's goals. For organizations operating large and extensive systems to ask hardware vendors to propose system designs, as is often done, is to transfer the most crucial systems decisions to a party least capable of making them.

Medium range: The user

1. *User Interaction.* The ability to anticipate user requirements and developments rather than simply react to them, depends, to a large degree, on a sophisticated understanding of the user business and data processing environment by the data communications staff. (Of course, the same is true for the other subsystem specialties, but because of its nature, the data communications subsystem is usually the most dynamic, and thus, needs to be more closely attuned to the user.) This understanding can only be obtained by the personal interaction of the data communications staff with users. Functional isolation and the lack of a broad user viewpoint are deadly to medium and long range planning.
2. *Full Costing of Teleprocessing Systems.* In a recent situation it was endlessly argued as to what the system response time should be, and it was simply arbitrarily set. Then it was noticed that operators paced their data input by the response time of input edit messages. Full cost analysis of the system showed that an improvement in the data communications network response time was paid many times over by the reduced operator costs associated with higher throughput per operator.

Long range planning

As previously mentioned, it makes little sense to talk about data communications long range planning, because data communications is just one subsystem of a larger data processing system. Effective long range planning must consider so many broad areas of uncertainty that it can only be effectively carried out on an overall systems basis.

In essence long range planning consists of projection and evaluation of possible future data processing architectural postures, given expected developments in: multiprocessing, limits of large processor architecture, operating systems, file management, multi-site operations, resource sharing networks, and business developments. Thus, long range planning for data communications (or any other subsystem) cannot be addressed out of the context of advanced planning for the data processing system in general. And data processing planning is only effective to the extent it is integrated with broad business planning.

As discussed under organizational considerations, ad hoc devices for crossing organizational boundaries are necessary to bring user and technician together for effective long range planning. A common error, however, is to assign the planning function to a special staff who then makes planning recommendations to operating managers. In my experience the success of a planning effort is inversely proportional to the autonomy of the planning staff. The value of planning to managers is more in their participation in the process than consumption of the product.

REFERENCES

1. Hopewell, L., "Trends in Data Communications," *Datamation*, August, 1973.
2. Withington, F. G., *The Organization of the Data Processing Function*, Wiley-Interscience, N.Y. 1972.
3. Thompson, J. D., *Organizations in Action*, McGraw-Hill, N.Y., 1967.
4. Chou, W. H. Frank and R. Van Slyke, "Simulation of Centralized Computer Communications Systems," *Proceedings of the 3rd Data Communications Conference*, Nov. 1973.
5. Ackoff, R. L., *A Concept of Corporate Planning*, Wiley Interscience, N.Y. 1970.
6. Hopewell, L., W. Chou, and Howard Frank, "Analysis of Architectural Strategies for a Large Message Switching Network: A Case Study," *IEEE Computer*, April, 1973.

Effective use of data communications hardware

by PATRICK McGREGOR

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

The broad range of perspectives of those involved with data communication networks makes effective use of hardware a topic of many dimensions. We will consider here the particular areas of cost, reliability, performance, and flexibility. Although these are not all-inclusive areas of network evaluation, they do encompass a sizeable portion of most evaluation efforts. Each of the areas is itself broad, and there is considerable overlapping. However, without a functional separation into such areas it is particularly difficult to approach the topic. We will try to note the interaction between the areas whenever possible.

The area of cost is perhaps the broadest, conceivably extending to a profit-loss statement of a network's operation. We will take a much narrower perspective of considering the line cost and purchase or rental cost of the hardware used to implement a network. Such costs are easily found, and consequently the cost of a particular network design implemented with a particular set of hardware is also easily found. However, it should be noted that the design of a network to minimize cost is often a very difficult task.

A factor of major importance to the successful operation of a network is its reliability. Reliability depends on two distinct aspects of the network; its components and its structure. We will be concerned primarily with the use of hardware alternatives to achieve reliable structures.

The performance of a network can be both difficult to define and difficult to evaluate. We will use the terminal response time as the performance measure, and queueing theory to appraise the effectiveness of hardware alternatives.

Flexibility is an area in which there may be no quantitative measures. The ability to expand a network in an orderly fashion, to adapt to new objectives, and to meet new requirements are all important. However, no one has yet quantified, at least very successfully, the impact of these capabilities. Consequently, only a qualitative discussion of flexibility is presented.

A thorough discussion of each of the above areas would require a book (or many books, e.g., books by James Martin). We make no such attempt here. Rather, it is our contention that significant insight into effective use of data communications hardware can be found through a functional approach to the discussion. It is this tack which we adopt.

A BASIC DATA COMMUNICATIONS NETWORK

In this paper we are concerned with the effective use of hardware in the implementation of a data communications network. An example of such a network is shown in Figure 1. The primary objective of the network is access by terminals to computers. As shown in Figure 1, such a network may involve more than one computer. We will be concerned with the terminal-to-computer aspects of the network, and consequently will restrict attention to single computer networks. An example of such a centralized computer network, and the basic hardware used for communications, is shown in Figure 2. In this example the CPU communicates with the terminals over leased, multidrop, voice-grade, full-duplex lines. Such lines represent only one of several alternatives for establishing a connection between a terminal and a computer. Other alternatives include connections on a dial basis, such as the DDD network and WATS service, leased low speed lines, leased high speed lines, TELEX and TWX, etc. Which alternative to use is intimately tied to cost and performance requirements of a specific case. However, in general leased lines are cost effective for terminals in use four or more hours per day. In most of the discussion which follows, we assume leased lines are appropriate.

Leased, multipoint, voice-grade, full-duplex lines are composed of trunks between central offices, and local loops connecting the CPU and the terminals to their respective central offices. The central office is the point where local loops and trunk lines are bridged to form the multipoint line. (We will not be looking at "daisy-chained" lines.) The cost of using such lines depends on a variety of factors, and is usually detailed in a legal document called a tariff.[1] A typical cost structure is shown in Table I, and will be used in later examples.

A voice-grade line may be roughly characterized as having useable bandwidth extending from 300 Hz to 3400 Hz.[2] Full duplex lines can transfer information simultaneously in both directions, while half duplex lines can transfer information in only one direction at a time. Both computers and terminals supply and accept information in the form of a digital baseband signal. The function of the modem (modulator-demodulator), as shown in Figure 3, is to interface the digital baseband requirement to the analogue bandpass requirement. A variety of techniques may be used to accom-

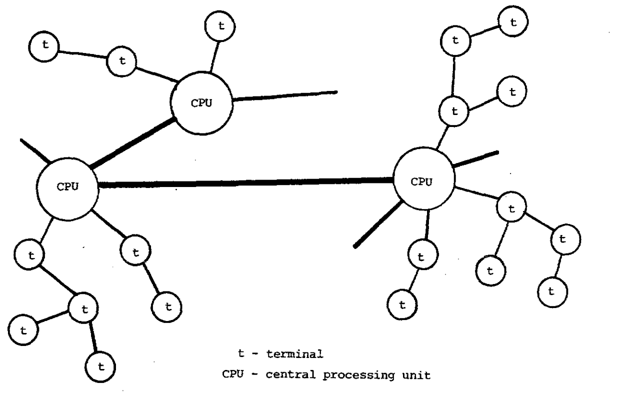


Figure 1—Example of a data communications network

plish the interfacing, giving rise to an extraordinary range of modem cost and performance characteristics.[3,4] Typical modem costs for various transmission rates are shown in Table II. We will use these costs in the examples which follow.

A network as described above, and portrayed in Figure 2, may be viewed as a basic structure for a data communications network. A wide range of hardware devices may make several other structures attractive. We consider alternative structures, and associated devices, in the following sections.

COST

There are several devices which may enhance the cost-effectiveness of a network. In this section we consider several such devices, and examine conditions for their use. We will start at the terminal and work in toward the computer.

Modem sharing unit

A modem sharing unit (MSU), or multiple access coupler, is a device for connecting several (typically up to six) terminals to a single modem. The terminals are usually restricted to be in the same location (within 50 feet of the MSU). Information received over the modem is "broadcast" to all terminals connected to the MSU, and the first terminal to appropriately respond gains access to the modem. Thus, in a polled terminal environment, the MSU is functionally transparent.

The purpose of an MSU is to replace the multiple local loops connecting the terminals in one location to the central office with only one local loop. The MSU effectively bridges the terminals onto the multidrop line at the local user's location rather than at the telephone company's central office. The cost of an MSU is approximately $600, with a rental rate of $25/month. With the typical modem costs noted earlier, it is cost effective to use an MSU for two or more terminals for any data rate. A location with four terminals each requiring a 2400 baud modem would result in savings of three modems plus line drops, minus the cost of the MSU, or 3×(50/modem+13.75/FDX drop)−25/MSU =$166.25/month.

TABLE I—Typical Cost Structure for Leased Voice-Grade Lines

MILEAGE CHARGE (HALF-DUPLEX)

First 25 Miles	(1-25)	$3.00/mile/mo.
Next 75 Miles	(26-100)	$2.10/mile/mo.
Next 150 Miles	(101-250)	$1.50/mile/mo.
Next 250 Miles	(251-500)	$1.05/mile/mo.
Additional	(500-over)	$ .75/mile/mo.

ADDITIVE FACTORS

First drop in exchange	$12.50/mo.
Additional station—same exchange	$ 7.50/mo.
Installation (one time charge)	$10.00

Full Duplex—10 percent additional for mileage charges and drops.

Multiplexer

We will use "facility" to refer to the part of the telephone plant described in terms of its properties as a transmission medium, and "channel" to refer to a functional communications path. A channel is described by its capacity, i.e., the maximum rate at which information can be acceptably transferred over it. A channel for the transfer of digital data is formed by placing a modem at each end of a facility. The capacity of the channel, or maximum data rate acceptable, depends on a variety of factors, including the bandwidth of

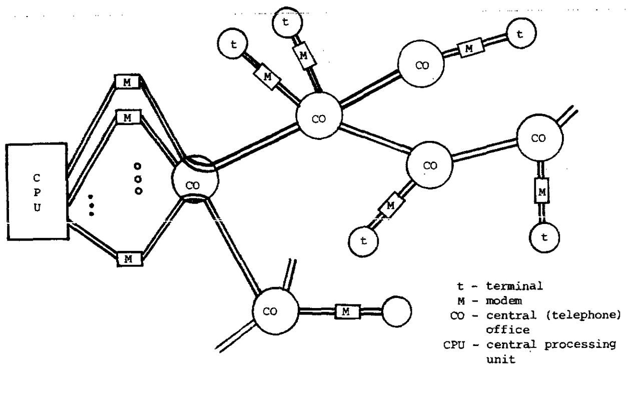

t
M
CO
CPU
t - terminal
M - modem
CO - central (telephone) office
CPU - central processing unit

Figure 2—Example of centralized network

TABLE II—Typical Modem Costs

Transmission Rate (bps)	Purchase ($)	Lease ($/mo.)
75	500	25
110	500	25
150	500	25
300	500	25
600	500	25
1,200	500	25
1,800	750	35
2,000	1,780	50
2,400	1,780	50
3,600	3,620	100
4,800	4,800	120
7,200	7,200	185
9,600	9,750	265

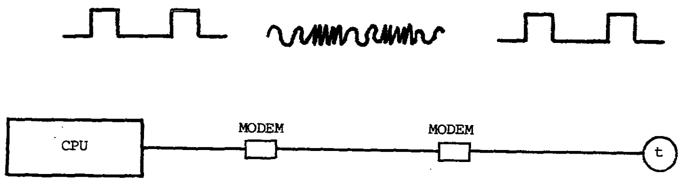


Figure 3—Function of modem

the facility and the hardware characteristics of the modems. The use of one facility to form several separate channels is called multiplexing. A device which combines multiple facilities, each used for one or more distinct channels, into one facility, formed into the same distinct channels, is called a multiplexer. A device performing the reverse process, i.e., transforming one facility, formed into several channels, into multiple facilities, each with one or more of the channels, is called a demultiplexer. Many current hardware devices perform multiplexing in one direction, and demultiplexing in the other direction. Such a device is usually simply called a multiplexer.

The channel is the functional communications path, whereas the facility is part of the hardware used to form a channel. A multiplexer does not alter the channel structure of the network, and thus is functionally transparent. However, the physical facilities from which channels are formed determine a large part of network costs. Multiplexing offers a way to achieve significant economies in facilities use. To understand these economies, it is helpful to examine the two fundamental approaches to implementing multiplexing.

One approach is to divide the bandwidth of the facility into several separate segments, and allow each segment to serve a separate channel. This is referred to as frequency division multiplexing (FDM), and is graphically portrayed in Figure 4. The second approach is to establish a high speed data stream over the facility and assign periodic time slots, or bit positions, of the data stream to separate channels. This is referred to as time division multiplexing (TDM), and is graphically portrayed in Figure 5. There are several variations on the implementation of these approaches.[5-9]

We turn now to the economics achievable by multiplexing. Consider two locations each containing three terminals to be connected to the computer, as shown in Figure 6. If each terminal is in almost constant use at 150 bps, a leased low speed line from each terminal to the computer may be used,

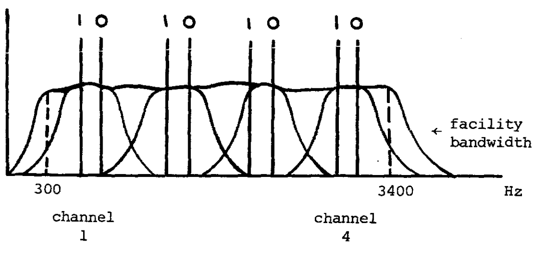


Figure 4—FDM

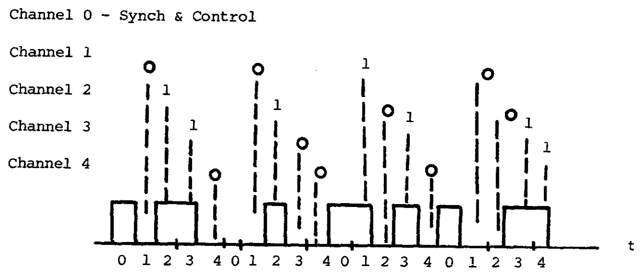


Figure 5—TDM

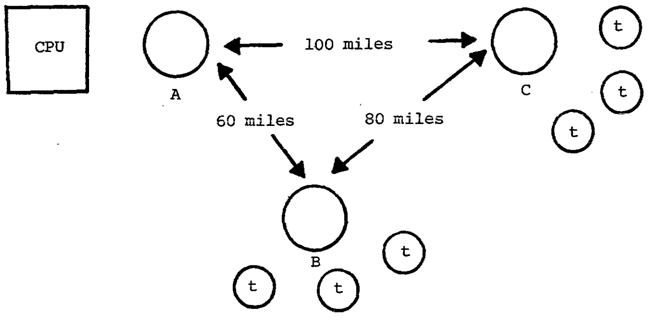


Figure 6—Six terminals in two offices

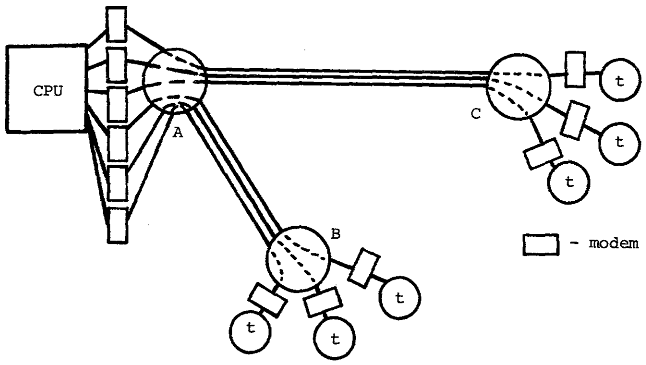


Figure 7—Six point-to-point lines

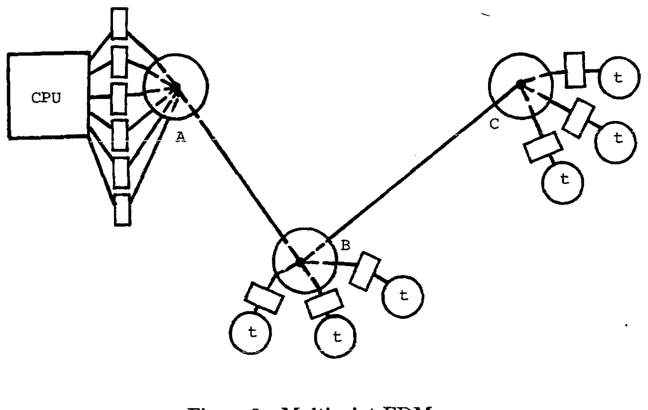


Figure 8—Multipoint FDM

(a) LOW SPEED POINT-POINT (FDX)

Modem Cost	12×$25/150 bps modem	$300.00
Mileage Cost	3×($1.93/mile×60)	347.40
	3×($1.93/mile×100)	579.00
First Stations	3×($34.38/station)	103.14
Additional Stations	6×($10.32/station)	61.92
		$1,391.46/mo

(b) VOICE GRADE MULTIPOINT (FDX)

Modem Cost	12×($25/150 bps modem)	$300.00
Mileage Cost	60 miles	
	$3.30/mile×25	82.50
	$2.31/mile×35	80.85
	80 miles	
	$3.30/mile×25	82.50
	$2.31/mile×55	127.05
First Stations	3×($13.75/station)	41.25
Additional Stations	6×($8.25/station)	49.50
		$763.65/mo.

$1,391.46/mo.−$763.65/mo.=$627.81/mo.

Figure 9—Cost analysis—FDM

as shown in Figure 7. Based on a typical low speed line tariff, the cost of such an arrangement is $1391.46/month, as detailed in Figure 9a. Alternatively, a multipoint voice grade facility may be used with each modem operating with a distinct pair of tones for its data transmission, as shown in Figure 8. The cost of this FDM approach is $763.65/month, as detailed in Figure 9b. This gives a very substantial savings of $627.81/month, or a reduction of about 45 percent of the point-to-point cost.

Current FDM techniques are able to achieve an aggregate bit rate of approximately 2000 bps on a voice grade facility, and are often implemented with devices having greater common circuitry at locations of multiple terminals than simply a separate modem per terminal, thus achieving even greater economies.: The cost per channel may be roughly estimated at $500 purchase, $25/month rental. Current modems can achieve a bit rate of 9600 bps over a voice grade facility. Through the use of digital logic, TDM systems can use this capability to almost full advantage, yielding a system with considerably greater bandwidth efficiency than one with FDM. The logic circuitry required for TDM gives it a different cost structure than for FDM, having a typical purchase price of $1000 plus $150 per channel per station, or $50/month plus $10/channel/month per station rental. A high speed modem is also required for each station. The current reductions in logic costs will serve to enhance the cost effectiveness of TDM systems relative to FDM systems.[10]

To examine the potential economy of TDM, we refer again to Figure 6. In this case assume each terminal is almost constantly in use at 1200 bps, eliminating the possibility of FDM multiplexing. If each terminal is connected directly to the computer via a leased voice grade line, as shown in Figure 7, the cost is $1648.05/month. By placing a TDM unit at each location, and multiplexing C to B, and B to A, with a single voice grade line used in each link, a cost of $1284.15/month is achieved, giving a savings of $363.90/month. Clearly substantially greater savings could be achieved in cases with more terminals per location and greater distances between locations.

In the above examples we have illustrated the use of multiplexing to permit several point-to-point lines to be replaced by one multipoint line. In the basic network portrayed in Figure 2, the multidrop line was used to supply a single channel accessed by each terminal on a polled basis. The maximum number of terminals which can be placed on such a channel depends on the capacity of the channel and the traffic generated by the terminals. It is easy to envision the potential benefit of multiplexers for this case by simply considering each terminal in the examples to be an end of a multipoint polled line. The actual cost differential would depend on the cost of connecting the multipoint lines to the computer without requiring them to go through the multiplexing locations. A significant question in this case is where to place the multipoint lines and multiplexers, a question beyond the scope of this paper.[11,12]

Concentrator

The word "concentration" appears to have a very broad meaning in data communications. We will discuss only one narrow interpretation of concentration.

Consider a device having several facilities connected to its input, and only one facility connected to its output. At this point the device may be a multiplexer. However, it is distinguished by the following characteristic: the single facility on the output side carries one channel, the capacity of which is less than the sum of all the capacities on its input side. Such a device providing effective communications is called a concentrator. A multiplexer is transparent to the channel structure of a network; a concentrator obviously is not.

The percent of time a channel is used is called its utilization. Many terminals generate data for transmission at an average rate which is much less than the capacity of the channel; resulting in channels with low utilization. A concentrator achieves economic advantage by replacing several low utilization channels with one highly utilized channel. A prerequisite for a concentrator is that its output channel capacity be greater than the sume of the average data rates of the terminals on its input. It is at this point perhaps helpful to examine the difference between a multiplexer and a concentrator in more detail.

To each time slot of each channel on the input of a TDM, a time slot is assigned in the high capacity channel on its output. This effectively divides the high capacity output channel into several separate subchannels, each associated with a particular channel on the input. It does not matter whether or not a time slot is being used to transfer information. A concentrator has more time slots arriving on its input side than leaving on its output side. Each time slot carrying information must be assigned a time slot on the output side. Thus a concentrator must be able to identify which time slots are in fact transferring information. Furthermore, it must be able to assign output time slots to this information in such a manner as to be understood by whatever device is on the other end of the output channel. Although the average number of time slots carrying information on the input will be less than the number available on the output, the random nature of terminal use may result in the number of slots carrying information arriving over a brief interval being greater than the number of slots available on the output. Hence, the concentrator must also have the ability to buffer the arriving information as it waits for available slots. The requirements of intelligence and storage for a concentrator invariably lead to its implementation with a minicomputer. The actual operation of concentrators varies considerably, but is usually much more sophisticated than the simple bit packing noted above.[5,13,14,15] By performing such local operations as polling, error checking, line control, etc., and transferring information to the computer with efficient high speed transmission techniques, the concentrator can achieve an apparent output channel utilization in excess of 100 percent.

The minicomputer implementation of a concentrator implies a fundamental component cost of approximately $10,000. Compared to a $1000 cost of a multiplexer, such a figure requires large economies to be achieved for cost effectiveness. Concentrators can typically handle 100 channels, (provided reasonable traffic characteristics). However, hardware required in addition to the minicomputer to achieve this capability raises the cost to approximately $20,000, or $500/month rental (excluding maintenance).

As an example of the economies achievable by a concentrator, consider a situation where 300 terminals on the west coast are to be connected to a computer in Chicago. The terminals are assumed to be interactive CRT's requiring a channel capacity of 1200 bps. Up to four terminals can share a channel in a polled fashion. Thus 75 channels are required from the LA vicinity to Chicago to implement the system with multiplexers. This requires approximately 10 fully equipped multiplexers at $2200 each ($1000+8×$150/channel), or the same approximate cost as a concentrator. However, with a concentrator reducing the required channel capacity by at least 30 percent, the line cost reductions would be on the order of $10,000/month.

Biplexers

A biplexer is a device which uses two voice grade lines to effectively achieve a single high speed channel (up to 19.2 kbps). Such a device must be able to compensate for the possible differential delays of the two separate facilities. Typically, acceptable operation can be achieved with the two lines diversely routed with differential delays up to ½ second.

The cost effectiveness of a biplexer is principally derived from the current tariff structure for high speed lines versus voice grade lines. Although in general the cost of capacity is a concave function, the quantized nature of offerings by common carriers can make two low capacity channels more cost effective than one high capacity channel. A 19.2 kbps channel for 500 miles would cost $3187/month (based on prorating the use of a 48 KC facility). Assuming a $250/month rental for a 19.2 kbps modems, such a channel would cost $3687/month. Two voice grade lines would cost only $1660/month. The biplexer has a typical cost of $5000, or $175/month rental, and would require four 9600 bps modems. Thus, the use of a biplexer and two voice grade lines would cost $3070/month, a net savings of more than $600/month. If the 19.2 kbps could not be prorated, the savings would be dramatically more.

A typical application of a biplexer would be to achieve a high speed channel from a concentrator to the central computer. Although the functions of the biplexer could be performed in software, the low cost of the biplexer hardware and its off loading effect on the concentrator often makes the biplexer the cost-effective alternative.

Port sharing unit

A port sharing unit (PSU) is a device for connecting several (typically up to six) modems to a single computer port (or concentrator, or multiplexer). The PSU broadcasts data from the port to all the modems, and delivers data to the port from the first modem to generate an appropriate response. Thus, in an environment of several simultaneously polled multipoint lines terminating at the computer, the PSU is functionally transparent. In general, cost-effectiveness is achieved by using only one multipoint line rather than several simultaneously polled lines, thereby saving local loop costs and port costs. However, occasionally telephone plant performance results in an operational restriction on the number of points acceptable on a single multipoint line. Similarly, reliability considerations may also require multiple lines. In such circumstances, a PSU may find effective use.

A PSU costs approximately $600, with a rental rate of $25/month. Its cost effectiveness is easily appraised for computers with additional ports achieved by the addition of hardware. However, the cost structure of ports for computers are so varied that we will not venture a quantitative example.

Front end processors

The central computer and terminals use the data communications network to interchange information. The general facility of a computer for transferring information between it and the outside world is its input/output (I/O) channel. Particular devices are connected with a hardware interface. In the case of a communications line, the modem terminating the line must be interfaced with the CPU. The overhead required for a large CPU to interact with many communications lines at a modem level is far too great to be economically attractive. Thus a sophisticated interface is used to handle the modem interaction, and only useful in-

TABLE III—Reliability Values

Device	Percent Inoperative (PF)	Percent Operative (P)
Modem (M)	.001	.999
MSU	.001	.999
PSU	.001	.999
CPU	.020	.980
Long Lines (LL)	.003	.997
Local Loop (LOC)	.002	.998
Terminal	.007	.993
Multiplexer (MX)	.001	.999
Concentrator (C)	.005	.995
FEP	.010	.990
LTU	.002	.998

formation is transferred through the I/O channel to the CPU. In the early history of such interfaces, hardwired logic devices called Line Termination Units (LTU) were used. More recently, it has become very attractive to use minicomputers to accomplish this task. Such minicomputers are called Front End Processors (FEP).

The software capabilities of minicomputers results in a very broad range of sophistication in their use as FEPs.[16,17,14,13] Consequently, the cost effectiveness of an FEP is difficult to quantitatively define. The costs of FEPs range over the costs of minicomputer systems, i.e., from \$6,000 upward. The cost of an FEP to be used with a large CPU in a large network may be expected to start at \$50,000 purchase, or a monthly rental of approximately \$1000. Such an FEP may have acceptable performance for up to 100 lines. In general, minicomputers have been found cost effective even in a strict emulation of hardwired LTUs.[18] We will consider the impact of their use on performance in a later section.

RELIABILITY

The reliability of a data communication network may be characterized in a great many ways.[19,20,21] We will consider two measures of reliability of particular use in characterizing centralized networks:

(1) the expected percent time a terminal cannot communicate with the CPU, and
(2) the expected percent time an entire office cannot communicate with the CPU.

The first criterion reflects the reduction in average network throughput due to failures, while the second is an indication of the network's ability to withstand catastrophic failures. The expected percent time a terminal cannot communicate with the CPU is the same as the probability a terminal cannot communicate with the CPU, and the fraction of terminals unable to communicate with the CPU. We will use the various terminologies interchangeably.

The reliability of a network is dependent on two fundamentally distinct factors: the hardware component reliability and the structural properties of the network. The vendors of hardware have historically been conscious of the reliability aspects of their products, (although not always concerned). Unfortunately, many networks are being designed today with little consciousness on the part of the designer of the reliability impact of the network structure. In this section we examine the effective use of hardware to achieve reliable network structures.

In order to quantitatively appraise the reliability of a particular network, it is first necessary to know the reliability of the hardware elements of the network. Although various manufacturers define their device reliability in various ways, and some not at all, to use the network reliability measures described above it is appropriate to define device reliability in terms of probability of failure. Such probabilities may be interpreted as the average down time of a device compared to the period intended for its active use. Different devices from different manufacturers are sure to have different failure rates. The probabilities of failure assumed in this paper are shown in Table III. These values are based primarily on observations of an existing network. They will serve to show the relative impact of structural variations on network reliability. Note that the probability of operation is simply one minus the probability of failure. Note also that reliability will be dominated by the CPU and terminals. We will not consider changing these devices.

A simple network

Perhaps the simplest network structure to analyze in terms of reliability is a leased line from each office to the CPU, with each terminal in the office bridged onto the line with a local loop, as shown in Figure 10. Since there is no redundancy on the path between the CPU and the terminal, all components on the path must be operative in order for communications to take place. Thus, the probability of a terminal being unable to communicate with the CPU is simply one minus the product of the probabilities that the individual components are all operative, or

$$PF_{T\text{-}CPU} = 1 - (P_{CPU} \times P_{MOD} \times P_{LOC} \times P_{LL} \times P_{LOC} \times P_{MOD} \times P_{T}) = .036$$

The probability that an office of k terminals cannot communicate with the CPU is the probability that all k terminals are disconnected, or

$$PF_{OFFICE\text{-}CPU} = 1 - P_{CPU} \times P_{MOD} \times P_{LOC} \times P_{LL} \times [1 - (1 - P_{LOC} \times P_{MOD} \times P_{T})^{k}] = .026$$

With k greater than one, the term within the brackets is, for all practical purposes, equal to one. We now consider

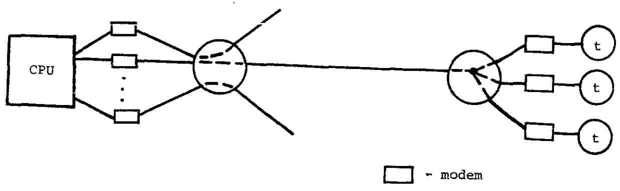


Figure 10—A simple network

the effective use of hardware for reliability in more complex structures.

Line redundancy

A simple means of improving network reliability is line redundancy. The probability of a concurrent failure of two disjoint leased lines is .000009, sufficiently small as to be taken as zero in comparison to other network component failure rates. The probability of a concurrent failure of two disjoint channels (modem+local loop+leased line+local loop +modem) is .0000804, again negligible. Thus provision of redundancy can achieve almost perfect reliability in comparison to the CPU and terminal. However, the cost of such redundancy can be enormous. We will consider cost effective hardware for providing near perfect communications reliability. Note that redundant modems, local loops, and leased lines gives a reliability of $PF_{T\text{-}CPU}=.02686$ and $PF_{OFFICE\text{-}CPU}=.02000$, with a doubling of network costs. We will use these figures as representing a "near perfect" network.

A modem sharing unit (MSU) can be used to reduce costs of a network by reducing the number of modems and local loops required for each office. In Figure 11 such an arrangement to achieve a more economical redundancy is shown. In this case the reliability measures are

$$PF_{T\text{-}CPU}=1-P_{CPU}\times[1-(1-P_{MOD}\times P_{LOC}\times P_{LL}\times P_{LOC}\times P_{MOD}\times P_{MSU})^2]\times P_T=.02696$$

$$PF_{OFFICE\text{-}CPU}=1-P_{CPU}\times[1-(1-P_{MOD}\times P_{LOC}\times P_{LL}\times P_{LOC}\times P_{MOD}\times P_{MSU})^2]=.02010$$

Virtually the same reliability is achieved as with total redundancy, but with a cost differential of replacing a pair of modems and local loops for each terminal in an office with only two modems, local loops, and MSU's for the entire office.

The major expense in redundancy is usually the redundant leased line. This expense can be greatly reduced by replacing the leased line with a dial-up capability at the office. In this case, the dotted line in Figure 11 would represent a dialed connection. The additional cost of a dial option on the modems is negligible compared to the line cost saved, and again the reliability of total redundancy is virtually achieved. There may be a convenience factor of having to establish the dialed connection at times of failure, but not necessarily so if the computer can identify failed conditions and initiate an automatic dial. Note that the use of MSU's at the office may result in a cost reduction sufficient to make this almost

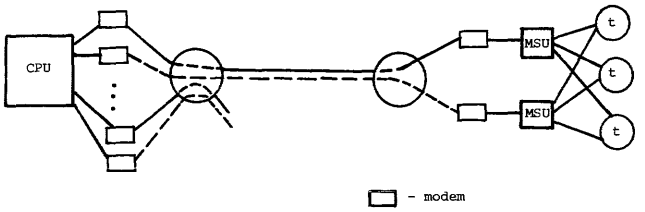


Figure 11—Use of MSU for redundancy

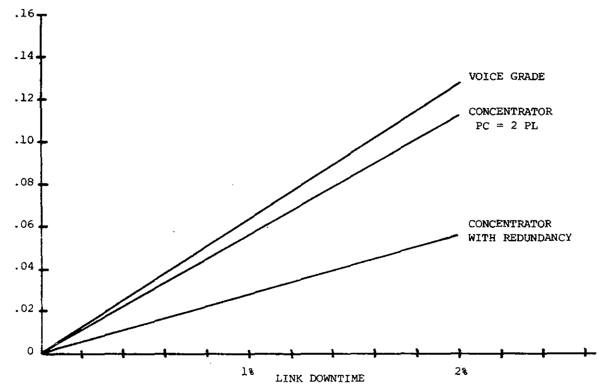


Figure 12

total-redundancy reliable system less costly than the non-redundant basic network described earlier. Further cost reduction may be achieved with negligible loss in reliability by the use of a port sharing unit at the CPU.

Finally, consider the use of a single MSU and modem, with the modem having dial back-up capability. In this case only line redundancy is provided, but with negligible cost. The reliability is then

$$PF_{T\text{-}CPU}=1-P_{CPU}\times P_{MOD}\times P_{MOD}\times P_{MSU}\times P_T=.029777$$

$$PF_{OFFICE\text{-}CPU}=1-P_{CPU}\times P_{MOD}\times P_{MOD}\times P_{MSU}=.022937$$

where we have left out the line terms due to their "perfect" reliability. Although this is clearly not as reliable as total-redundancy, it is a significant improvement over the non-redundancy case, as is achieved at negligible cost. In fact, in comparison to the basic network described earlier, it achieves greater reliability with considerable cost reduction.

Other configurations are possible for various degrees of cost-reliability trade-offs. However, the above considerations are sufficient to point up the feasibility of obtaining highly reliable networks while retaining cost-effectiveness, (and cost-effectiveness while retaining reliability, too).

Multipoint lines, multiplexers, and concentrators

The failure rate of any particular leased line depends on a multitude of factors, many of which are determined by the telephone company. The assumption of a uniform probability of failure for all lines, such as done above, reflects the designer's lack of knowledge, and control, of the network implementation. However, as shown above, such an assumption is useful in considering reliability aspects of the network. Similarly, for multipoint lines it is useful to consider all line segments as independent entities with uniform probabilities of failure. With such an assumption, the probability of a particular office (or terminal) not being able to communicate with the computer depends on the office's position on the multipoint line. Although this complicates calculation of reliability, it still can be readily done, by algorithm, with a computer. Results of such calculations for a typical network are shown in Figure 12 as a function of the multipoint segment reliability.[19,20]

Concentrators achieve economies by replacing several long, low utilization, multipoint lines (trees) centered at the CPU with many short, low utilization trees centered at the concentrator, and one high utilization line from the concentrator to the CPU. The average number of links between a terminal and the CPU is much smaller in a design using concentrators, and therefore a perfectly reliable concentrator should improve reliability. In fact, even moderately unreliable concentrators typically improve reliability, as illustrated in Figure 12, with the probability of a concentrator failure being twice that of a link failure. This is another case of cost-effective hardware being also reliability effective.

Since there are generally a large number of terminals connected to the CPU through a concentrator, further significant improvement in network reliability can be achieved by increasing the reliability of the concentrators and concentrator-to-CPU lines. As before, an additional leased line or dial back-up can achieve almost perfect reliability in the concentrator-to-CPU line. To make the concentrators almost perfectly reliable, a redundant concentrator can be placed at each site. A less expensive, but equally effective approach, may be to place a multiplexer at each concentrator site, and a demultiplexer at any two concentrator (or CPU) sites. A failed concentrator then has its load absorbed by one of the two selected concentrators. The improvement in reliability achieved by such redundancy is shown in Figure 12. The number of concentrators and concentrator-to-CPU lines is small compared to the number of terminals, and thus the reliability improvement achievable by such redundancy can be obtained without appreciable increase in total network cost.

Multiplexers are usually much simpler devices than concentrators, with correspondingly greater reliability, less cost, and less dramatic impact on the network. In general, the greater the number of channels multiplexed, the greater the impact of multiplexer failure on reliability. However, the economies of multiplexers is usually such as to retain cost effectiveness even with redundancy.

In general, network reliability is improved at the expense of redundant components. As shown above, often cost-effective hardware can achieve sufficient economies to allow redundancy, thereby improving reliability, while still maintaining significant network cost advantages.

PERFORMANCE

There are many possible measures for the performance of a data communications network.[22] We will examine the impact of data communications hardware on one of the most common of these measures, the terminal response time.

The terminal response time may be loosely defined as the length of time between the instant a terminal is ready to send a message to the CPU until the instant the response of the CPU is completely received at the terminal. The precise definition of "message" and "response" depends on the particular case considered. In the discussion which follows, the appropriate interpretation will be clear from context.

The performance of a network depends on a great many factors, including hardware capabilities, overhead traffic, line speeds, message length distributions, message arrival distributions, buffer sizes, etc. Appropriately modeling and analyzing a network to predict terminal response time is an extraordinarily difficult task, usually involving considerable queueing theory and often simulation.[23,24] However, it is possible to gain substantial insights by considering some of the fundamental characteristics of the networks. It is this approach that we take.

Terminal demands

The Model 33 Teletypewriter has enjoyed unparalleled success in being selected for use in interactive networks, perhaps due to its low cost, availability, and hard copy output. It operates at low speed (110 bps) on an asynchronous character-by-character basis. The market now has several keyboard/printer terminals of similar characteristics.[25] In small networks, the terminals may be connected directly to the CPU, with each character typed resulting in an interrupt of CPU processing for terminal servicing. In such systems the terminal response time may be taken as the delay before the echo of the character by the CPU is printed on the terminal. However, in time sharing systems the user usually interprets the system's performance on the basis of the delay between the typing of a "carriage return", signifying the end of a line, and the return of a "line feed", signifying acceptance of the line by the CPU. It is this measure which we will use here.

As the number of terminals serviced by a CPU becomes large, and the level of terminal activity increases, the processing of incoming traffic on a character interrupt basis leads to intolerable overhead. Consequently, it is often attractive to alter the network to a multipoint, polled, buffered form in which the CPU directs terminals as to when to send information, and terminals are buffered so that multiple character messages may be accumulated before requiring transmission. This not only serves to reduce CPU overhead, but also increases line efficiency.[26] The upgrading of the network may be accomplished in a variety of ways, three of which we note below.

First, there is the option of upgrading each terminal with a buffer and stunt box. This has the attractions of simplicity, equipment retention, and cost. When more than one terminal is present at an office, there is the possibility of enhancing cost effectiveness by sharing the buffer and intelligence among all terminals through a Terminal Control Unit (TCU). Such a unit may be a hardwired logic device or a very simple minicomputer or microcomputer. The latter becomes more attractive in view of the potential for intelligence expansion. Finally, it may be attractive to simply replace terminals with more sophisticated terminals having not only buffering and polling response capabilities, but also better display approaches, faster printing, etc. Depending on the particular case, each of these alternatives may appear most attractive, and some combination of all may be most appropriate for some networks.

Queueing delays

In a data communication network it is often the case that the greatest portion of response time is the time spent by a message in storage, waiting for transmission or processing. For this reason, queueing theory has been a particularly effective tool in gaining insight into network performance.

A data communication network can often be modeled as an interconnection of queues, with the terminal response time being the sum of the waiting times of a message in each of the queues. For a single server queue with Poisson arrivals, any distribution of service times, and any service discipline (provided not dependent on service time), the average waiting time of a message from the instant of its arrival to the instant of its departure is given by the Polloczek-Khintchine equation:[27]

$$W = s + \frac{\lambda(s^2 + \sigma_s^2)}{2(1 - \lambda s)}$$

where

λ—average number of arrivals per second
s—average service time of a customer-seconds
σ_s^2—variance of the service time distribution

The percent time the server is busy servicing some customer is defined as the utilization, $\rho = \lambda s$. The dependence of the average waiting time on utilization and the variance of the service time is shown in Figure 13.

The first place where a message may experience delay is in waiting to be transmitted on a polled multipoint line. As noted above, the service discipline does not affect the average waiting time, and thus the cyclic discipline of a polled line may still be modeled as a single server queue. However, an accurate model must give considerable attention to an appropriate definition of service time, to include not only the message transmission time, but also the negative polling response times, and the polling interrupts for transmission of messages from the CPU to the terminals (this latter may in fact dominate performance). The appropriate modeling and analysis is indeed complex.[28] However, inspection of Figure 13 gives some pertinent results. First, for highly utilized lines ($\rho > .7$), a little reduction in utilization can give a large reduction in waiting time. Second, for lowly utilized lines ($\rho < .5$), the waiting time is dominated by the service time, and substantial improvement occurs only with substantial reduction in service time. There are numerous ways in which reductions can be made in utilization and service time. We consider here two fundamental approaches: (1) increase transmission rate and (2) reduce the number of terminals on a line.

In many cases the modem is a stand alone unit, and an increase in transmission rate may be accomplished with a simple exchange of modems. The impact of such an increase extends beyond the simple reduction of time required to transmit a message. The time required for a negative polling operation is also reduced, and thus the time required for a polling cycle is reduced. Similarly, the percent time the line is used to transmit messages from the CPU to terminals is also reduced. Thus, increase in transmission rate may be expected to have a greater than linear impact on service time and utilization.

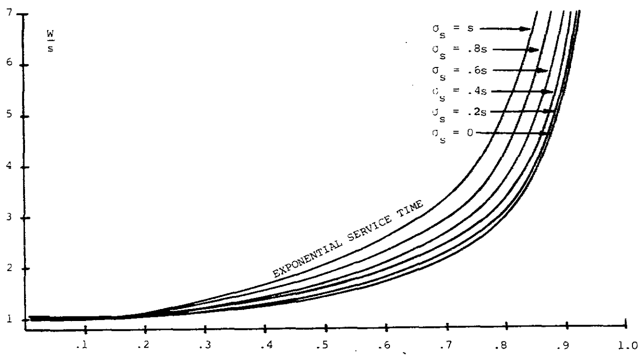


Figure 13

Reducing the number of terminals on a line is accomplished by replacing one multipoint line with more than one line. When the terminals have built in modems, this approach may be particularly attractive. Like increasing transmission rate, reducing the number of terminals also has a greater than linear impact on service time and utilization. Not only is the number of terminals polled reduced, thereby shortening the polling cycle, but the average number of messages arriving at the line is reduced, and correspondingly, the average number of transmissions from the CPU to terminals on a line is reduced.

Both increasing transmission rate and reducing the number of terminals imply an increase in cost, the usual trade-off for performance. However, as noted earlier, cost effective use of hardware can minimize this trade-off. In particular, the increase in transmission rate from low speeds ($\leq$300 bps) to medium speeds (600, 1200 bps) can occasionally cost only the incidental exchange cost, with no monthly rental increase (provided no change in tariff occurs). When a line is replaced with more than one line, it may become cost effective to use multiplexers, or even concentrators. However, it should be noted that TDM's introduce an incidental transmission delay (½ bit for bit interleaved and ½ character for character interleaved[7]), and concentrators introduce another queue in the network.

CPU bound

The heart of the centralized network is the CPU. It is the resource to which the network is intended to permit access. The CPU itself is often modeled as a network of queues.[29] For our purposes it is sufficient to envision it as a single server queue for which the service time and utilization depend not only on the processing time of a message, but also on the overhead required to service the communication lines.

The efficiency of large CPU's is particularly vulnerable to the inefficiencies of slow I/O devices, unscheduled demands, and, although less so, to the overhead required for small processing tasks. Consequently, manufacturers have provided an extension to the CPU to handle terminal communications, permitting interruption of the CPU only when significant information has been accumulated. Initially, such devices were hardwired logic units, called Line Control Units, and performed the tasks of:[30]

- Assembling received bits into characters
- Buffering
- Inserts and strips control characters
- Validates received data
- Transmits characters to terminals

More recently it has become cost effective to use a minicomputer to perform these tasks.[18] Furthermore, performance can be considerably improved by using the parallel processing capability of the minicomputer to further off-load the CPU by performing the tasks of:

- Assembling characters into messages
- Message validation
- Line and terminal control
- Codes, formats, and speed conversions
- Message transmission to terminals

Minicomputers serving to control communications for the CPU are called Front End Processors (FEP), and vary considerably in cost and capabilities.[13]

In a large centralized network, even the FEP may be overloaded by the time consuming tasks of polling, message assembly, code conversion, etc. In such cases it may be necessary to distribute the communications overhead tasks throughout the network by assigning such tasks to concentrators and/or terminal control units. The use of minicomputers for this purpose also permits another degree of off-loading of the CPU by performing simple processing functions, such as accounting, program assembly, program editing, etc.[31]

FLEXIBILITY

We interpret the term flexibility as both a vertical and horizontal capability of the network. Vertical in the sense of growth to handle more terminals and heavier traffic. Horizontal in the sense of handling a broader variety of terminals. The two key words for such flexibility are software and modularity.

The advent of minicomputers has provided a cost effective means of providing a software capability at various levels of the network. Minicomputers have generally shown themselves to be cost-effective devices for performing the primary tasks associated with terminal control units, concentrators, and front end processors. However, the programmability of minicomputers may give an even greater advantage to their use in terms of flexibility, both vertical and horizontal.

Both vertical and horizontal flexibility is found in the use of a minicomputer as a terminal control unit with its potential for easily performing the following tasks:

- simultaneously handling a variety of terminal codes, and converting the codes to a standard form,
- simultaneously handling a variety of terminal protocols,
- interacting with terminals on a character basis while interacting with a concentrator, FEP, or CPU on a message basis,
- providing sophisticated error control procedures in message transmission,
- locally handling terminal failures,
- providing local accounting and simple processing tasks,
- providing local traffic control.

The software nature of minicomputers and the modular nature of minicomputer systems makes extension of TCU in any of the above areas a relatively simple task, and can be accomplished with little effect on the CPU and other components of the network. This aspect of isolation of modifications required for new terminals, procedures, etc., to a simple network component, greatly enhances the cost-effectiveness of flexibility.

The use of a minicomputer as a concentrator has the same potential for flexibility as a TCU, with the same attractions of ease of modification and isolation of modification from the CPU. In addition, the software capability of a concentrator also gives a potential for easily performing the following tasks:

- performing the multidrop polling,
- high speed data transmission techniques and sophisticated error control in communication with the CPU or FEP,
- handling mixed terminal speeds on the same multidrop line,
- handling different line protocols, and in particular, interacting with terminals on some lines and more sophisticated TCU's on other lines,
- local handling of line failures.

Both terminal control units and concentrators achieve flexibility by cost-effective distribution of intelligence in the network. Minicomputer systems tend to be of a modular nature in both hardware and software, permitting easy growth in both horizontal and vertical flexibility. The use of a minicomputer as a front end processor differs from its use as a concentrator to the extent that its mission is to achieve economic advantage by improving CPU efficiency rather than by reducing line costs. In reality, the operation of a concentrator and FEP may be very similar, with the FEP performing the same tasks as the concentrator in order to efficiently utilize the I/O channel of the CPU. However, where much of the flexibility of a concentrator is an added attraction, it is a necessity for the FEP.

Modularity is the ability to upgrade a system by the addition of appropriate modules to an already existing fundamental structure. Minicomputers usually have both software and hardware modularity, and consequently offer tremendous flexibility with their use in a network. Other devices, such as multiplexers, can also be modular in structure. The tradeoff for such modularity has traditionally been cost and efficiency, due to the expense and inefficiency of a general structure to permit growth. However, the achievements of cost reductions in hardware, and in particular logic, has made such a trade-off far less dramatic than before. The advantages in terms of the potential for a smooth growth process in general far outweigh the current cost advantages achievable in a non-modular approach.

CONCLUSION

There are many ways to effectively use data communications hardware. Modem sharing units can be used to reduce the number of local loops and modems required, multiplexers to reduce the number of lines required, and concentrators to reduce the number of channels required. Redundancy is the key to reliability, but need not be particularly expensive when achieved with cost-effective hardware. Improved performance is achieved by off-loading the CPU, higher transmission rates, and fewer terminals per line. The key words for flexibility are software and modularity. There is much more to be said than space permits. There are other devices to be considered, and other structures to be compared. The technology of devices and structures is rapidly advancing. It is our hope that the discussion presented above will help in evaluating the impact of such advances.

REFERENCES

1. Gerla, M., "New Line Tariffs and Their Impact on Network Design," *Proceedings of NCC Conference*, Chicago, Illinois, May 6-10, 1974.
2. Martin, J., *Teleprocessing Network Organization*, Prentice-Hall, Englewood Cliffs, New Jersey, 1970.
3. Davey, J. R., "Modems," *Proceedings of the IEEE*, Vol. 60, No. 11, November 1972, pp. 1284-1292.
4. Davis, S., "Modems: Their Operating Principles and Applications," *Computer Design*, September 1973, pp. 75-83.
5. Doll, D. R., "Multiplexing and Concentration," *Proceedings of the IEEE*, Vol. 60, No. 11, November 1972, pp. 1313-1321.
6. Fleig, W. E., "A Stuffing TDM for Independent T1 Bit Streams," *Telecommunications*, July 1972, pp. 23-32.
7. Glasgal, R., "Advanced Concepts in Time Division Multiplexing," *Telecommunications*, October 1972, pp. 27-32.
8. Libby, P. T., "Time Division Multiplexing," *Telecommunications*, June 1972, pp. 55-60.
9. Pack, C. D., "The Effects of Multiplexing on a Computer-Communications System," *Communications of the ACM*, Vol. 16, No. 3, March 1973, pp. 161-168.
10. Parker, D. W., "Multiplexer Selection for Effective Data Communications," *Telecommunications*, January 1973, pp. 21-27.
11. Chou, W. and A. Kershenbaum, "Unified Algorithm for Designing Multidrop Teleprocessing Networks," Data Networks Analysis and Design, *Third Data Communications Symposium*, St. Petersburg, Florida, November 13-15, 1973.
12. Woo, L. S. and D. T. Tang, "Optimization of Teleprocessing Networks with Concentrators," *National Telecommunications Conference*, Atlanta, Georgia, November 26-28, 1973.
13. Newport, C. B. and J. Ryzlak, "Communication Processors," *Proceedings of the IEEE*, Vol. 60, No. 11, November 1972, pp. 1321-1332.
14. Mills, D. L., "Communication Software," *Proceedings of the IEEE*, Vol. 60, No. 11, November 1972, pp. 1333-1341.
15. Starks, J. P., "Functional Systems Approach to Remote Message Concentrator Design," *Computer Design*, March 1973, pp. 61-64.
16. Feinroth, Y., E. Franceschini and M. Goldstein, "Telecommunications Using a Front-End Minicomputer," *Communications of the ACM*, March 1973, Vol. 16, No. 3, pp. 153-160.
17. Hibbs, R. B., "Features of an Advanced Front-End CPU," *Proceedings of the Spring-Joint Computer Conference*, 1971, pp. 15-21.
18. Pryke, J. T. M., "A Front-End Primer for IBM Users," *Datamation*, April 1973, pp. 46-50.
19. Frank, H., "Providing Reliable Networks with Unreliable Components," Data Networks: Analysis and Design, *Third Data Communications Symposium*, St. Petersburg, Florida, November 13-15, 1973, pp. 161-164.
20. Frank, H. and R. Van Slyke, "Reliability Considerations in the Growth of Computer Communication Networks," *National Telecommunications Conference*, Atlanta, Georgia, November 26-28, 1973, pp. 2201-2205.
21. Wilkov, R. S., "Analysis and Design of Reliable Computer Networks," *IEEE Trans. on Communications*, June 1972, pp. 660-678.
22. Meister, B., H. R. Muller and H. R. Rudin, "New Optimization Criteria for Message-Switching Networks," *IEEE Trans. on Communications Technology*, June 1971, pp. 256-260.
23. Chou, W., H. Frank and R. Van Slyke, "Simulation of Centralized Computer Communications Systems," Data Networks: Analysis and Design, *Third Data Communications Symposium*, St. Petersburg, Florida, November 13-15, 1973, pp. 121-130.
24. Van Slyke, R., W. Chou and H. Frank, "Avoiding Simulation in Simulation Computer Communication Networks," *Proceedings of National Computer Conference*, June 4-8, 1973, pp. 165-169.
25. Hobbs, L. C., "Modems," *Proceedings of the IEEE*, Vol. 60, No. 11, November 1972, pp. 1273-1284.
26. Martin, J., *Design of Real Time Computer Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1967.
27. Cox, D. R. and W. L. Smith, *Queues*, Metheun and Company, London, England, 1961.
28. Konheim, A. G. and B. Meister, "Polling in a Multidrop Communication System's Waiting Line Analysis," *ACM/IEEE Second Symposium on Problems in the Optimization of Data Communication Systems*, Palo Alto, California, October 20-22, 1971, pp. 124-129.
29. Buzen, J. P., "Queueing Network Models of Multiprogramming," Deputy for Command and Management Systems, HQ Electronic Systems Division (AFSC), L. G. Hanscom Field, Bedford, Massachusetts, *Publ. #ESD-TR-71-345*, August 1971.
30. Audin, G., "TCU!, Front End! Switch! What are Their Functions?," Modern Data, June 1973, p. 47.
31. Bouknight, W. J., G. R. Grossman and D. M. Grothe, "The ARPA Network Terminal System—A New Approach to Network Access," Data Networks: Analysis and Design, *Third Data Communications Symposium*, St. Petersburg, Florida, November 13-15, 1973, pp. 73-79.

New line tariffs and their impact on network design

by MARIO GERLA

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

The advances in computer and communication technology have made available several new techniques (packet-switching, satellite multiple access, loops, packet radio, etc.) for communication network design which, in addition to the traditional techniques (point-to-point, multidrop, line-switching, etc.) provide a large gamut of alternatives for network users. At the same time, the increasing competition between common and specialized carriers is making available to the user a variety of leased and switched, voice and data communication services with different qualities and different rates. The user who wants to implement a private communication network is therefore faced with a variety of possible alternatives, corresponding to different techniques and line tariffs.

In this paper, we assume that the communication technique has been selected and limit our investigations to the impact of the various line tariff alternatives on network design.

Network design programs which take into account several tariff alternatives should have a very modular structure, with separate functions implemented in separate modules, so that tariff changes would affect only a small number of modules. More precisely, the network designer should be able to subdivide, on the basis of line tariffs, the various design problems and subproblems into a small number of classes, each class corresponding to a different cost structure. For each cost structure, a different design module is developed, which is general enough to handle all problems and tariffs that correspond to such cost structure. The global design program is obtained by combining the proper modules; for instance, the program for a two-level satellite network would require two tariff modules: one corresponding to the satellite cost structure and one corresponding to the cost structure of the terrestrial subnetworks.

In the following sections, we introduce a possible cost structure classification and discuss some very general techniques for the solution of network design problems within each class.

PRESENT AND PROPOSED LINE TARIFFS

Several companies are presently providing, or applying for authorization to provide, communication facilities and services in the U.S.A. Line tariffs vary according to company, type of service, and quality of service. A detailed description of tariffs is beyond the scope of the paper. Here, we analyze the parameters which are most important for network design (i.e., channel bandwidth, dependence of line cost on mileage, volume discount, etc.) and establish criteria for the comparison and classification of line tariffs.

The following is a list of the most significant present or proposed tariffs for dedicated voice or data channels.

A. *Common Carriers* (Bell System, Independent Telephone Companies, Western Union, etc.)

1. Narrowband: for transmission of data up to 300 bps. Line cost depends on distance only. The following are interexchange mileage charges for channel type 1001, full duplex:

Miles	\$/Mile×Month
1-100	1.10
101-250	0.77
251-500	0.44
501-1000	0.33
1001-up	0.22

Notice that per mile charge decreases with distance.

2. Voiceband: for transmission of voice and data up to 9,600 bps. Line charge is the sum of the mileage and service terminal charges. The mileage charge depends only on the distance between the two exchanges. The service terminal charge corresponds to the local circuit from exchange to user and is independent of distance. The following are the tariffs for type 3000 full duplex voiceband channels. Interexchange mileage charge:

Miles	\$/Mile×Month
1-25	3.30
26-100	2.31
101-250	1.65
251-500	1.155
501-up	0.825

Service terminal charge: 16.50 \$/Month; installation charge: 50 \$. When data are transmitted on the voice band channel, the use of modems is required. The cost of modems varies according to transmission rate (from .3 up to 9.6 kbs).

3. Series 8000: Accommodates 12 voice channels or data up to 48 kbs. Total cost is given by mileage and service terminal costs. The service terminal cost is 425 $/Month. The mileage cost is as follows:

Miles	$/Mile×Month
0-250	15.00
251-500	10.50
501-up	7.50

4. Series 5000 (Telpak): type 5700 (formerly Telpak C) accommodates 60 voice channels or a 240 kbs data rate; type 5800 (formerly Telpak D) accommodates 240 voice channels or a 1,000 kbs data rate. For data transmission, the total bandwidth can be divided into subchannels of the desired bandwidth. Mileage charges are as follows:

Type	$/Mile×Month
5700	30.00
5800	85.00

Notice that there is no mileage rate discount based on distance.
Service terminal charges are as follows:
Type 5703 (19.2 kbs or 6 voice ch.): 425 $/Mo.
Type 5701 (50 kbs or 12 voice ch.): 425 $/Mo.
Type 5751 (230 kbs or 60 voice ch.): 625 $/Mo.

5. High-Low density tariff (proposed by AT&T): Approximately 370 locations are defined to be high density points; the remaining are low density points. For a half duplex channel the following interexchange mileage rates apply:

High point-high point:	.85 $/Mile×Mo.
High point-low point or low point-low point:	2.50 $/Mile×Mo.
Short haul ($\leq$25 miles):	3.00 $/Mile×Mo.

Monthly channel terminal charges are $35 for Hi and $15 for Low; station terminal charges are $25 for both Hi and Low.
A low to low connection can be implemented either directly (in which case the low to low direct distance tariff applies) or through two intermediate high density points (in which case different tariffs apply to different segments).

6. Digital Data Service (DDS): this new data service, being developed and proposed for FCC approval by AT&T, is based on the T1 digital carrier network. Better quality and considerable economy can be obtained by transmitting data on T1 rather than on traditional analog channels. DDS will interconnect initially 24 major cities, and will be progressively extended to most of the 370 high density locations mentioned previously. Different channel bandwidths can be leased; the proposed rates are as follows:

Bandwidth (kbs)	Mileage charge ($/Mile×Mo.)	Service Terminal ($/Mo.)
2.4	.45	140
4.8	.60	200
9.6	.90	280
56.0	4.50	500

From a remote location a customer can access the DDS network via private analog channels of proper bandwidth and characteristics (series 3000, 5000 or 8000) and with adequate modems.

B. *Specialized Common Carriers* (Datran, MCI, etc.)

1. Datran: the network under development will offer digital communication service between district offices located in 35 major cities from the East to the West Coast. Users within 50 miles of each district office area can be connected to the network with Datran facilities; however, intradistrict rates will be different from interdistrict rates. The latter are expected to be similar to the rates offered by DDS.
2. MCI: both analog and digital service are presently offered connecting Chicago, St. Louis and several intermediate locations with future plans to include East and West Coast locations. Channels of bandwidth up to 1,000 kbs are available. The total line charge is the sum of intercity mileage charge, system access charge, and channel termination charge. The intercity charge is proportional to the distance between the two terminal cities. Intercity rates for some sample full duplex channels are as follows:

Bandwidth (kbs)	$/Mile×Mo.
4.8	1.32
9.6	1.50
50.0	14.90
240.0	23.00

The system access charge corresponds to the connection between city exchange and user location. Some sample rates of access charge for a user located in the metropolitan area around a city exchange follow:

Bandwidth (kbs)	$/Month.
4.8	51.
9.6	65.
50.0	300.
240.0	425.

A user located outside the metropolitan area must purchase the access channel from AT&T or the local telephone company. The channel termination charge differs from analog to digital and applies to any channel termination. Some sample rates are as follows:

Bandwidth (kbs)	Analog ($/Mo.)	Digital ($/Mo.)
4.8	10.	70.
9.6	12.	100.
50.0	40.	100.
230.0	82.	112.

C. *Value Added Networks* (VAN's): VAN's are communication service companies which lease transmission facilities from common or specialized carriers and resell communication services not available from the above (packet-switching, lower error rate, speed and code conversion, etc.) At this writing, only PCI has been granted FCC approval among the several VAN applicants (Graphnet, MCI, Data Transfer Corp., Telenet etc). Proposed tariffs are not available yet; however, there is indication that, at least for packet-switching networks, the rates will be independent of distance, and proportional to traffic volume (packets/sec.)

D. *Satellite Communications*: Several companies (WU, Amersat, CML, RCA, GTE, AT&T, etc.) have been granted (or are waiting for) FCC approval to sell private satellite communication services in the U.S.A. Most satellite carriers will provide, in addition to the satellite channel, a terrestrial backbone network to facilitate satellite access and improve overall reliability. Although most satellite tariffs have not yet been defined, it is anticipated that the total line cost will be given by the sum of the satellite and terrestrial cost components. In particular, the satellite segment (from antenna to antenna) will be much less expensive than a coast to coast terrestrial channel (e.g., a full duplex 56 kbs channel on the satellite will cost about 500 $/mo). As for the terrestrial cost, different rates will apply, depending on whether the customer provides his own ground stations, arranges for his own terrestrial access to the company's ground station, or finally uses the company's terrestrial network. Two general characteristics of satellite rates will be:

(1) Rates not too sensitive to distance, (2) Strong volume discount with respect to satellite bandwidth.

A GENERAL LINE COST CLASSIFICATION

In the leasing of communication facilities, the user is faced with a variety of alternatives which differ in cost, quality of transmission, delay, etc. In order to achieve a minimum cost network design, all such alternatives must be carefully considered. It is practically impossible to develop computer programs for network design which would take into account all the available commercial offerings. The best approach is to classify such offerings into a limited number of very general cost structures and to develop efficient algorithms for each structure. Specific problems can be solved by properly varying the input parameters of each algorithm.

Among the line cost characteristics that most often affect network design are: (1) dependence of line cost on end point locations and (2) volume discount that can be obtained by leasing channels of larger bandwidth. Based on such characteristics we identify three classes of line cost structures: (1) Distance Dependent (DID) structures: (2) Location Dependent (LOD) structures; and (3) Volume Discount (VOD) structures. The following is a description of the three classes, with some examples.

A. *DID Structures*: The cost *per channel* (or per unit bandwidth) from point A to point B is a function of distance (A, B) only. It is independent of the specific locations of A & B, and of the number of channels (or bandwidth) from A to B. Examples of DID Structures are the narrowband tariff, the interexchange tariff for type 3000 voiceband channels and the type 8000 tariff. If we restrict A and B to belong to a privileged set of points, then the HiLo density tariff (where A and B are high density points) and some of the VAN tariffs (where A and B are nodes of the value added network) can also be considered as DID Structures. In the latter case the cost per unit bandwidth might be even independent of distance.

B. *LOD Structures:* The cost per channel (or per unit bandwidth) from A to B depends on the specific locations of A and B. It is independent, however, of the number of channels (or bandwidth) from A to B (no volume discount). A typical example of an LOD structure is the HiLo density tariff. When either A or B or both are low density points, the rate depends not only on distance (A,B) but also on the geographical position of A and B with respect to high density points. Another example is offered by the VAN's, where the relative location of A and B with respect to network nodes clearly effects the connection cost. In problems where only one channel of given capacity must be allocated between A and B (and therefore volume discount does not apply) also DDS tariff and in general all specialized and satellite carrier tariffs can be considered as LOD structures: in fact, the cost of the channel will depend on the relative position of points A and B with respect to the DDS network, the special carrier network or the terrestrial backbone network of the satellite company.

C. *VOD Structures:* The cost of leasing *an additional channel* (or additional bandwidth) from A to B decreases with the number of channels (or bandwidth) already leased from A to B. Furthermore, the cost depends on the distance between A and B (but does not depend on their specific locations). Examples of VOD Structure are: the Telpak tariffs (where A and B can be any location in the U.S.A.); the DDS tariff (where A & B belong to the DDS network); the specialized carriers and satellite companies (where A & B belong to the respective networks).

A more general VOD structure would include line costs specifically dependent on terminal locations (as in LOD). However, we show in the next section that this more complex

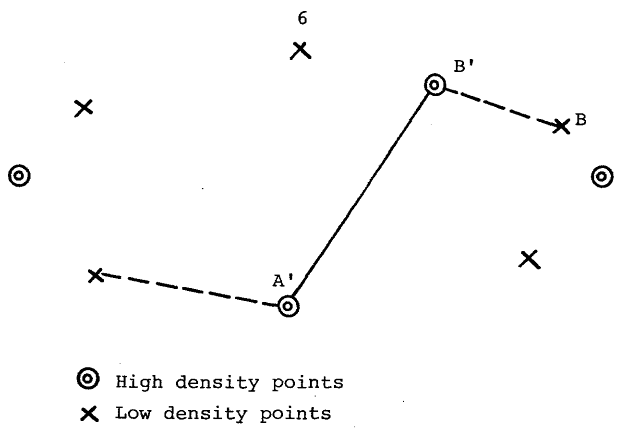


Figure 1—Minimum cost connection under HiLo structure

class of problems can often be reduced, by means of clustering and partitioning techniques, to the study of two separate problems of LOD and VOD type respectively.

LINE COST COMPUTATION FOR NETWORK DESIGN

A very general design problem formulation is as follows:

Given:

1. Traffic requirements (voice channels, data rates, etc.) between the various communication centers (computers, terminals, telephone exchanges, communication processors, etc.).
2. Scheme of implementation (point-to-point connections; packet-switching; line switching; multidrop; loops, etc.).
3. Available tariffs for lines and equipment.

Minimize:

Total communication cost (sum of line, termination, modem and communication processor costs).

Such That:

The traffic demand is accommodated with the required grade of performance (average delay, grade of service, reliability etc.).

Network design algorithms depend very critically on the particular cost structure corresponding to the available line tariffs.

In fact, we recall that the backbone of most network design algorithms is a minimal spanning tree (MST) computation,[1] or a shortest route (SR) computation,[2,3] or both.[4] In any case, both MST and SR computations require the notion of cost per channel (or cost per unit of bandwidth) associated with each node pair in the network. Furthermore, at some iteration of the algorithms, links are introduced or deleted according to cost criteria.[1,2] The nature of such costs, and their efficient evaluation have a very important impact on the overall efficiency of the design algorithm.

In the following we discuss some basic techniques for the design of networks with DID, LOD, and VOD structures respectively.

A. *DID Structures:* For small network size, one can compute, using direct distance, the costs of all potential circuits between all node* pairs and store the information in a cost matrix. All subsequent operations (MST, SR, link insertion or removal) are based on such information. Most centralized network algorithms presently available use this approach. For networks with thousands of nodes, however, the full cost matrix cannot be stored in core: to overcome the difficulty, a reduced cost matrix, which contains the costs of potential links from each node to only a subset of the nodes, might be considered. The criterion for selecting such subsets is highly critical to the efficiency of the overall algorithm, and varies from problem to problem. An example of this technique is discussed in Reference 1.

B. *LOD Structures:* An LOD structure can be reduced to a DID structure upon computation of circuit costs between all (or a subset of) node pairs. In general, the cost between two nodes in an LOD structure is defined as the sum of the costs of the circuits along the minimum cost route. More precisely, if A and B is an arbitrary node pair, and A' and B' are any two points between which a special tariff exists (e.g., A' and B' are high density points, or switching centers of a DDS network etc.), the cost $C(A, B)$ between A and B is defined as (see Figure 1):

$$C(A,B)=$$
$$\min\{C_d(A,B), \min_{\forall A',B'} [C_d(A,A')+C_s(A',B')+C_d(B',B)]\} \quad (1)$$

Where:

$C_d(X,Y)$ = direct distance cost of a circuit from X to Y

$C_s(X,Y)$ = special tariff cost of a circuit from X to Y

A straightforward way of obtaining all the new costs is as follows: consider an augmented network, which includes original nodes and special tariff nodes, and compute the matrix of minimum costs between all node pairs of the augmented network, using the standard shortest route algorithms.[5] Such a minimum cost matrix provides all the desired new costs. Core and computer time requirements, however, make the method inefficient for large networks; in such cases it is possible to improve efficiency by considering costs between each node and a subset only of the total set of nodes, in a way very similar to that described for DID structures. A powerful algorithm, based on the latter technique, has been recently developed for the high low density tariff: in particu-

* In a communication network, "node" is any device that terminates a circuit and, eventually, provides connections between several circuits (e.g. terminal, concentrator, telephone exchange, satellite ground station, communication processor, etc.).

lar it was found that on the average we need only to consider the cost from each low density point to four high density points.[6]

An interesting example of LOD cost structure is offered by a network design that can utilize unsaturated links of a preexisting Telpak network. On such links the cost of adding a new channel (marginal cost) is zero. The minimum cost for establishing a voice grade link between any two telpak nodes is given by the cost of the shortest path between such nodes, where link length is defined as the marginal link cost. If we assume that the new network requirements are small enough so that no Telpak link becomes saturated, the preexisting Telpak network induces an LOD cost structure where the shortest route, and therefore the cost of a circuit between any two points, depends on their relative locations with respect to Telpak nodes.

C. *VOD Structures:* Under such cost structures, network design tends to route the external requirements on the routes that provide the best economy of scale (which are not necessarily the most direct routes). Among the several problems with a VOD structure are: the Telpak problem;[4] the design of the long haul common carrier network;[3] and the design of a packet-switching network.[2] The network design algorithms proposed for the three above examples and described in the corresponding references have the following common feature: they repeatedly compute shortest routes according to appropriate link costs (which change from iteration to iteration) and redistribute (or "deviate") requirements on such routes. The effect of such deviations is to achieve better economy of scale and, therefore, reduce cost.

Without entering into details, we simply mention that the efficiency of the VOD algorithms is strictly related to the nature of the cost-capacity functions of the links. For example, if such functions are continuous and concave (or convex), it is possible to develop efficient, suboptimal and, in some cases, optimal algorithms using the theory of mathematical programming.[2] In most practical applications, however, channel capacities are available for a discrete set of values only, and therefore, cost-capacity functions are not continuous. In such cases, various degrees of heuristics must be used, and the difficulty of the problem depends on the nature of the functions. In particular, link functions which are rather irregular, "neither concave nor convex," and which have large capacity jumps (like the Telpak case shown in Figure 2) are difficult to handle during network design. On the other hand, link functions with small capacity jumps and which can be reasonably approximated by continuous curves (one such example is provided by MCI's tariff FCC #1 (see Figure 3)) lead to quite efficient algorithms.

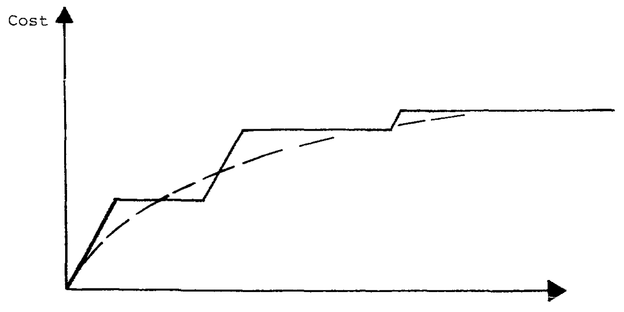


Figure 2—Telpak-like tariff

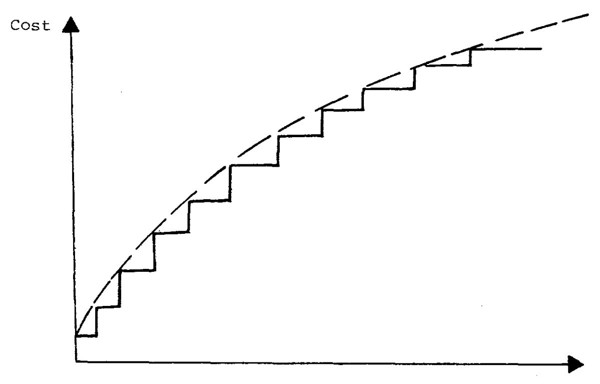


Figure 3—MCI-like tariff

As a general rule, although optimal topology will depend on many factors (economy of scale, throughput level, node geographical location, traffic requirements, etc.), it is possible to anticipate that cost structures with strong economy of scale lead to tree topologies, while structures with mild economy of scale lead to highly connected topologies.

D. *Partitioning and Node Location Problems:* Network partitioning consists of dividing the nodes into subsets and solving a separate design problem for each subset. This operation typically requires the solution of a location problem, because partitions are connected to each other, or to a central node, through one or more "exchange" nodes whose locations must be optimally selected (e.g. concentrator location, for centralized networks, or satellite ground station location, for satellite networks).

Network partitioning is necessary for the design of networks that are too large to be handled in their entirety by computer algorithms. Furthermore, network partitioning is the natural consequence of a VOD cost structure or, more generally, of any economy of scale situation where it pays to implement a multi-level hierarchical structure. In a two-level structure, where several regional networks are connected by one national network, interregional traffic is sent from regional nodes to their respective regional exchanges, and from there through the national network. National links carry a high traffic volume and can achieve a better volume discount than can regional links.

A typical example of partitioning and node location problem is the design of a two-level network where the national network contains both terrestrial and satellite links, with private ground stations. The national net shows a strong economy of scale effect, due to the volume discount for satellite bandwidth and, more important, to the very high cost of ground stations. In order to take advantage of such an economy of scale, we would like to send as much traffic as possible through the satellite links.

However, there is a complex trade off between terrestrial line cost, satellite bandwidth cost, and ground station cost. A reasonable heuristic approach is as follows: we fix arbitrarily the initial number and location of ground stations; we design a low cost national topology; and we assign to the global problem an LOD cost structure (where line cost between two nodes of the national network is considerably less than direct distance cost). Then, we accommodate the node pair requirements of the global network along minimum cost routes relative to the LOD structure. At the same time, we determine national network requirements and regional partitions. Next, using such requirements, we design the national network, which has a VOD cost structure. Once the national solution is obtained, we recompute the LOD cost structure, based on marginal costs, and reiterate the procedure. The algorithm also provides for relocation, addition, or deletion of ground stations.

CONCLUSION

The advances in communication technology and the increased competition between communication services vendors will generate, as a consequence, a variety of offerings and of tariffs which will be subject to more frequent changes than before. The user, in implementing and upgrading his communications network, will be faced with a variety of alternatives, and will be able to achieve considerable economies by carefully selecting the most appropriate communication facility. In particular, the user will be asked to: (1) specify, in more precise terms than before, his communication requirements (traffic volume, traffic statistics, grade of service, admissible delay, error rate, reliability, etc.) so that such requirements can be met at minimum cost by the use of proper technology and proper communication offerings; (2) be prepared to reconfigure his network more often than before in order to take advantage of rapidly changing cost and quality of services.

As for the impact of new tariffs and services on communication network design, we can anticipate that:

1. *All Network Design Will Be Computerized:* In fact, it will be too cumbersome, if not impossible, to evaluate manually the trade offs between all the available alternatives.
2. *Programs Will Be More Modularized:* Each function (e.g., subnetwork design, partitioning, LOD and VOD cost computations, etc.) will be confined in a specific module so that tariff changes require only the modification of a few modules. Furthermore, new design algorithms can be easily built from existing modules.
3. *Network Design Will Be More Flexible:* in view of technology and tariff changes, network design must allow efficient reconfiguration of the sections affected by those changes.
4. *Network Performance Will Be More Frequently Monitored:* From traffic statistics and performance measurements, and on the basis of the new technology and tariffs, the communications manager must be able to determine when the network must be upgraded or reconfigured.
5. *Design Programs Will Be More Interactive:* Changes in technology, traffic and tariffs will require frequent verification that network design is still optimal or near optimal. In order to do that, one must be able to evaluate, in an interactive fashion, the sensitivity of network performance to changes of input parameters and design variables.

REFERENCES

1. Chou, W. and A. Kershenbaum, "A Unified Algorithm for Designing Multidrop Teleprocessing Networks," *3rd Data Communications Symposium Proceedings*, Tampa, Florida, November 1973.
2. Gerla, M., *The Design of Store-and-Forward Networks for Computer Communications*, School of Engineering and Applied Science, University of California, Los Angeles. Ph.D. Dissertation. January 1973.
3. Yaged, B. Jr., "Minimum Cost Routing for Static Network Models," *Networks*, 1: 139-172, 1971.
4. Goldstein, M., "Design of Long Distance Telecommunication Networks: the Telpak Problem," Report R-4, Office of Emergency Preparedness, January 1971.
5. Floyd, R. W., "Algorithm 97, Shortest Path," *Communications of the ACM*, 5 (6): 345, June 1962.
6. Private communication from Aaron Kerschenbaum, Network Analysis Corporation, Glen Cove, N.Y.

Tools for planning and designing data communications networks

by AARON KERSHENBAUM

Network Analysis Corporation
Glen Cove, New York

INTRODUCTION

The planning, design, and management of a large telecommunications network is a complex process involving many aspects of the network's cost and performance. Throughput capability, reliability, response time performance, and cost must all be considered in evaluating alternatives. For a large network, obtaining an accurate analysis of any one of these aspects is a formidable task involving cumbersome calculation and many decisions. The problem of obtaining an accurate analysis of the entire system is magnified by the fact that these individual factors interact strongly with one another. Thus, in order to design a "good" network, or verify that an existing network is operating properly, one must not only be able to solve each of the above problems, but also be able to evaluate the solution of each one in the context of solutions to the others.

Computer programs, if they are carefully designed and properly used, can be powerful tools in the planning and management process. While one cannot reasonably expect that such programs will solve all problems or even that they will solve any one problem perfectly, such programs can take a large burden off the analyst by providing accurate and reliable answers to subproblems which can then be used to answer global questions about the overall network.

Thus, these programs are used by the qualified analyst as tools. If the man-machine interface is good, the analyst and the computer together can solve problems that neither could solve as well separately and each works on the aspects of the overall problem which he can handle best. The analyst decomposes the overall problem into subproblems and couches his questions to the computer in such a way as to reflect the peculiarities of each particular network and the relative importance of each design criterion, which also varies from system to system. The machine answers specific questions in the context posed and enables the analyst to consider many more alternatives than he would otherwise have the time or inclination to approach using hand analysis alone.

This paper functionally describes programs which can be used as design and analysis tools, and the uses they can be put to. Examples of the benefits obtainable through the use of such tools in the design and analysis of actual networks are presented. Although much of the material presented can be applied to the design and analysis of all types of networks, the discussion is limited to centralized telecommunications networks.

PROGRAM ARCHITECTURE

There are many programs currently in use as network design and analysis tools,[1-8] and each approaches the overall problem from its own point of view. Some stress a comprehensive data base to describe the system. Others stress topological design, equipment selection, or some other single aspect in the design/analysis process. Still others approach the problem as a whole, solving many aspects of the problem simultaneously. Each of these approaches has its advantages, but most stop short of the ultimate goal of being true design tools as defined above.

Useful design tools must be reliable and flexible. If the user cannot be reasonably sure that the program will consistently produce accurate results which he can rely on, he might just as well solve the problem by hand. Similarly, it might be easier to solve a problem than it is to twist the problem into the form which will fit into a rigid program.

One way of achieving these goals is to build tools modularly, using a group of programs which are each designed for a specific purpose. Each module is simple to design, flexible, and reliable because it deals only with one particular aspect of the overall problem. It is tailored to that specific subproblem and derives its utility from being able to treat that subproblem well, rather than having to approximate a solution to many problems at once. A screwdriver is a reliable tool which can be used to put screws into a wide variety of objects, but consider the problem of designing a single tool which will cut wood, drive nails, and put in screws. Worse yet, consider the problem of having to work with such a tool.

Building tools modularly has additional advantages. Changes to one module to accommodate new equipment or tariffs need not affect other modules and thus, can be made with relative ease. This is another dimension of flexibility. Modules designed to solve a specific problem usually run efficiently and thus, many more alternatives can be considered that could be with a large, cumbersome program. This is another dimension of reliability.

These modules must, of course, be coordinated with one another as the solution of the global problem depends upon each individual subproblem solution. More so, the subproblem solutions often depend upon one another. This task falls mainly upon the analyst. He may choose to coordinate the modules by specifying input parameters to each module, by examining outputs, by interacting with the modules during their execution, or even by using a program manager module to control some of the flow of information among individual modules. In each case however, it is the analyst who decomposes the original problem, specifies the context of the solution of each subproblem, and assembles the solution of the global problem from its component parts. This places the responsibility upon the analyst of not only understanding the problem, but also of understanding what his tools can do for him and what they can not. The situation which thus arises is more realistic than one where this responsibility is placed on the tools and is a significant improvement over working with no tools at all.

INPUT MODULE AND DATA BASE

Before one can make use of design and analysis tools, he must first specify the network and his objectives to the programs he is using. While this is not the most conceptually difficult task in the design/analysis process, it can be the most time consuming unless it is done efficiently. This can best be done using an input module and a data base which interact with one another.

The input module reads in user supplied data including parameters and pointers into the data base. It then fills in the defaults and fetches user requested information from the data base, collecting all specified data in one place. Next it checks the collected data for completeness, correctness, and consistency, outputting error messages, and possibly halting if any sufficiently serious errors are encountered. Finally, it organizes the data into the structural form required by the other modules, and if the input module is run as a separate program, outputs this data structure to be read in by other modules.

The ideal mode of input is the interactive mode where the user is permitted to enter data while the program is running. Error checking and corrections can be done immediately, and the user can specify some parameters after having seen output from some of the modules. An important property of an input module running in an interactive mode is that parts of it will run after other modules are run or, in some cases, even while they are running. This differentiates it structurally from an input module run in a static mode, which usually does its whole job at the start of execution. Thus, an interactive input module should consist of several sections which are executable separately and closely linked to the modules controlling the overall program flow. For each particular network design, the task of inputting all the required data can be greatly simplified by incorporating an easily usable set of defaults, allowing the user to omit parameters and have the system fill them in. The NAMELIST facility, present in many higher level languages, which allows the user to input data in free format specifying both the variables' name and value and omitting variables which are to take default values, is one good way of implementing defaults.

The data base is a body of information which is not, in general, dependent upon a particular network (e.g., device characteristics or tariffs) and is necessary input to the program. The user can specify a block of information he wishes to make use of, and the input module fetches it from the data base thereby relieving the user of the burden of specifying each piece of information individually. One section of the input module, which in general may run entirely independently of the rest of the sections (and the rest of the program), is a procedure for creating and modifying the data base.

It may be necessary to input parameters which overwrite and supplement values gotten from the data base. A common example of this is pricing information which is rightfully part of the data base but is often varied from system to system because it is volume dependent. By providing this flexibility, the data base can be kept to a manageable size, excluding devices of marginal utility without fear of limiting the use of the program. It is also not necessary to constantly change the data base when a new line of equipment is introduced. Manual override parameters and system default parameters together provide the user with the ability to freely intermix input data with information gotten from the data base, using the program under a wide variety of circumstances with a minimum of effort.

NETWORK CONFIGURATION AND HARDWARE ALTERNATIVES

One of the first questions which must be answered in order to design a network is what hardware devices should be used in the design and how should the hardware be configured. The choice of central processor, peripherals, concentrators, multiplexers, terminal controllers, and terminals greatly influences the network's cost and performance, and forms the basis of all further analysis and evaluation of the system.

Central site configuration

Given a central processor (or several processors) and a set of peripheral devices (tapes, disks, drums, etc.), one must evaluate the throughput capability and cost effectiveness of the configuration. One can thus compare different lines of equipment and different configurations of a specific line of equipment.

A program of this nature can range from a simple closed-form queueing formula to a very elaborate brute-force simulation. The former approach, though simple, must adhere to rigid assumptions. Thus, the scope of its application is limited. The latter one requires enormous efforts both in developing and using the program, and is in general only useful for one particular system. Thus, it is quite impractical to use, as two essential features of a simulation program are that it be inexpensive for repetitive usage and versatile enough to be used in a varying operating environment. Therefore, this module should be a simulation program with embedded queueing models.

A properly constructed queueing model can represent many different systems by changing parameters and can accomplish many important tasks, such as considering systems containing peripheral devices with different characteristics, pinpointing traffic bottlenecks, and allowing arbitrary distributions for CPU processing time and message arrival rates.

Terminal configuration

The basic function of such a module is to evaluate the cost effectiveness and throughput capabilities of different types of terminals and terminal configurations. In particular, one may wish to evaluate the effectiveness of using terminal controllers with simple slave terminals as opposed to using more sophisticated terminals which can operate without controllers. For each configuration, the module calculates the number of terminals required as a function of throughput and the total cost versus throughput capacity.

Choice of multiplexers and concentrators

The decision whether or not to use concentrators and multiplexers in a network design must be partially based upon how much money can be saved in other parts of the network and how their inclusion will affect the network's reliability and response time performance. Configuration modules can help in making this decision by evaluating alternate devices and calculating the number of devices required and their cost for given traffic loads.

Each of these modules takes, as input, device characteristics and traffic characteristics and yields as output an evaluation of a hardware configuration. A final decision as to what hardware configuration should be used need not be made on the basis of such information alone, but instead, can be made using the output of other modules as well. By treating problems of hardware configuration in these modules however, rather than as part of a large program which considers other issues as well, one can obtain clear answers to these configuration questions and greatly simplify the tasks of the design and analysis modules.

RESPONSE TIME/THROUGHPUT ANALYSIS

One of the most important, and most difficult problems in analyzing a telecommunications network is getting an accurate analysis of response time and throughput capability. Without a clear picture of the system's response time performance, the analyst must make conservative assumptions during the design process, and runs the risk of introducing so much slack into the design that it cannot be sold in today's highly competitive market. Worse yet, he runs the risk of designing a system which will not work. When the system to be analyzed includes concentrators, multiplexers, terminal controllers, and complicated central processor hardware configurations, the problem becomes especially complex.

The output of the hardware configuration modules can be used to great advantage in solving this problem. One can explicitly simulate message flow along a single path between a terminal and the central computer by using the parameterized analytic queueing models in the configuration modules to represent system hardware and software. In this way, it is possible to accurately account for the delays introduced by hardware, software, and the presence of traffic from other lines incident to the path being simulated.

This approach to the problem is more effective than pure simulation or analytic modeling alone. Analytic models of the complex network of queues present in real systems can only be formulated by making simplifying assumptions which result in inaccuracies in the response times obtained from the model. Pure simulation, while more accurate, is cumbersome and results in a model which is specific to the particular system under consideration. Such models must be modified extensively for each new system considered. The hybrid technique, using analytic models for each device as part of a simulation model of the entire system, results in an accurate model which is at the same time flexible enough to be used in the analysis of a wide variety of different types of systems.

By means of input parameters, the user should be able to specify message length distributions for input and output messages, keying and display (or printing) times, distributions for message interarrival times, number of terminals (or terminal controllers) per line, number of lines per concentrator, line speeds, line discipline, polling and addressing sequence lengths, number of bits per character, propagation times, device turnaround times, message processing times, and parameters describing the delays in hardware devices. Additional input parameters controlling the operation of the simulation package might include multiple simulation option, trace feature, output control parameters, maximum simulated time, maximum number of transactions simulated, and transient response suppression.

The output of the response time/throughput module may be formatted in many ways, dependent upon the user's needs. Response times may be presented on an overall basis or components of response time such as waiting times in various devices may be presented separately if they are of interest. Response times may be given as functions of facility utilization (e.g., line utilization, or CPU occupancy). Empirical distributions may be presented if a detailed analysis of systemwide variation in response time is required, in particular if a constraint at a given percentile of traffic must be checked. In some applications, where leased lines are used, queueing delays may be of primary interest while in others, where contention or dialup is used, blocking probabilities may be required.

Thus, with the aid of output from the configuration modules, the response time/throughput module can produce an accurate and reliable analysis of a network even when it is complex and contains several levels of hardware. This analysis is not only interesting in its own right as it provides one of the most useful measures of a system's performance, but also, it can greatly simplify the task of the topological design module by providing it with useful relationships between response times and facility utilizations.

RELIABILITY ANALYSIS

One of the most neglected areas in the network design process is reliability analysis. Many networks are designed without giving any thought to this important measure of network performance, and only after painful experience with frequent failures and degraded response times following periods of downtime is it given belated consideration.

The most fundamental decisions in the configuration of a network—hardware selection, use of alternate paths to provide backup for critical communications lines, use of spare components for backup, and connection of on-line devices, in series or in parallel—relate strongly to the network's reliability. Here, as in the case of response time, inadequate analysis may lead to conservatism in design, or to excessive optimism, which can result in unacceptably poor system performance.

Given a topological design of a network and the failure rates of components (mean time to failure, mean time to recovery, or probability of failure) one must calculate reliability measures for the entire network. Here, as in the case of response time and throughput analysis, the output may be organized in different ways, depending upon the user's needs.

Usually, one wishes to obtain the average value of the percentage of time a terminal or location (with several terminals) can communicate with the central computer. Sometimes this information is useful on a terminal by terminal or location by location basis. One may also want the average reduction in system throughput capacity due to equipment failure.

If the network under consideration is topologically simple (i.e., a tree, a series parallel network, or a loop network), such reliability measures can be obtained analytically. A combination of analytic and simulation techniques is often highly effective in obtaining solutions in many practical situations which do not fall into any of these simple topologies. In particular, many networks can be analyzed as a collection of trees or loops which are interconnected by a more topologically complex backbone network. Each tree or loop can then be collapsed analytically into an equivalent node in the backbone network, and the problem can thus be reduced to that of a small number of nodes which can be analyzed using simulation. Alternatively, when the network under consideration is composed of many small topologically complex subnetworks linked by a backbone, each subnetwork can be separately reduced to an equivalent node in the backbone, and the backbone can then be analyzed efficiently. In each case, the network is collapsed from the extremities toward the center, replacing increasingly large subnetworks with equivalent nodes. If the overall network is not complex topologically, both the running time and memory requirements of the procedure need be only linearly proportional to the number of nodes in the network.

TOPOLOGICAL OPTIMIZATION

The task of topological optimization, i.e., deciding how to interconnect network locations as economically as possible while still meeting all performance constraints, is the most complex task facing the analyst, as it encompasses all the previously mentioned problems. The other modules at his disposal provide great assistance in this task; but he is still left with many decisions, especially if the network is large and geographically diverse enough to benefit from the economies offered by concentrators and multiplexers.

In order to design networks using concentrators and multiplexers, one must first choose the number of concentrators and multiplexers and their locations. Next, each terminal in the network must be associated with a particular concentrator or multiplexer. Then the routing and speed of the multidropped lines connecting the terminals to the concentrators and multiplexers must be decided upon. Finally, the routing and speed of the lines connecting the multiplexers and concentrators to the central computer must be decided.

A module which is to be a useful tool in solving these problems must itself be composed of submodules, each of which is designed for a specific purpose.

First, because of the variety of tariff structures currently in use, one needs a submodule for evaluating the cost of connecting any pair of points with a given speed line. Where certain tariff structures are applicable, such as the high/low density tariff, or where existing Telpaks can be used, intermediate points may be used in routing circuits, and this cost calculation may itself be complex. By separating the cost calculation from other functions, the topological optimization module can be used with a variety of tariff structures without becoming unduly complex.

The problems of associating terminals with concentrators and routing the multidrop lines connecting terminals to concentrators are separable from the other problems and should be treated by a separate submodule dedicated to this purpose. One such efficient algorithm has been the subject of another paper[4] by this author.

The remaining problems, which concern the location and interconnection of concentrators, can often be dealt with manually by the analyst, since the number of concentrator or multiplexer locations are often limited by the existence of manned facilities. If, despite these factors, the remaining problems are not tractable manually, a straightforward automated procedure can be implemented by embedding solutions to the multidrop line layout problem into an algorithm which evaluates different numbers of concentrators and different concentrator locations.

The topological optimization module requires, as input, information describing the applicable tariffs, traffic at each location, terminal locations, and utilization factors for lines and devices. In addition to this, concentrator and multiplexer locations must be input unless an automated procedure for their selection has been included. Notice, however, that performance criteria need not be input directly. It is the function of other modules to evaluate such criteria and produce as output the utilization factors and specific device types which are input here. This operation greatly simplifies the task of the module, allowing it to solve its problem accurately and efficiently.

COORDINATION OF THE MODULES

As has already been mentioned, the task of coordinating these modules usually falls on the analyst. Nonetheless, it is possible to lighten this burden to some extent by designing another module, the Program Manager, to assist the analyst in the performance of part of this task. The point must be stressed, however, that an attempt to use such a module as a substitute for insight on the part of the analyst can easily complicate the Program Manager and all the other modules, and seriously impair their functions.

The basic function of the Program Manager is to link the other modules together while they are executing, allowing them to pass information to one another which would help them in performing their individual tasks.

Some of the parameters passed to each module as input can be gotten from the output of other modules. Thus, the utilization constraints on lines and devices input to the topological optimization module (TOM) can be gotten directly from the output of the response time module (RTM) rather than having the user specify them as input. This not only makes the TOM easier to use; it also provides for greater flexibility in specifying these constraints, since the RTM can generate combinations of constraints that are too complex for a user to specify by hand. The RTM may also pass several different combinations of constraints and allow the TOM to choose the best set.

Similarly, the TOM can pass detailed information about a particular design to the RTM, allowing the RTM to produce a more exact analysis of that particular design than could be provided using only general parameters describing the network. The RTM could then reevaluate constraints and pass them back to the TOM, and the entire process can iteratively improve the network design.

A detailed description of alternate CPU and terminal configurations can be passed to the RTM, enabling it to produce a more exact response time and throughput analysis of the network using each configuration. The results of this analysis can then be passed back to the configuration modules, allowing them to produce a more detailed comparison of the alternatives. Again, this process can be iterated. Similarly, the topology of a design can be passed directly from the TOM to the reliability evaluation module. Note, however, that in each case, the functions of the module is kept distinct, even though each module makes use of information provided by other modules. This is as close as one can reasonably expect to come to having one's cake and eating it too.

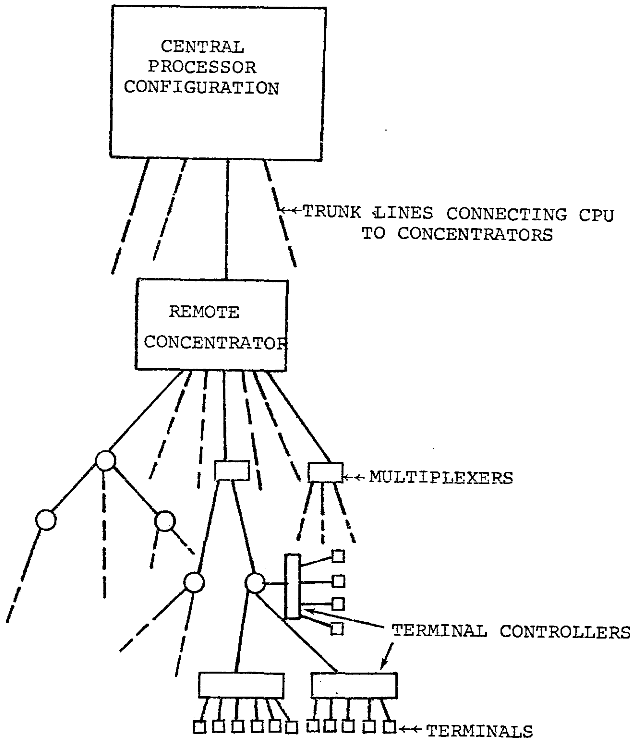

CENTRAL PROCESSOR CONFIGURATION
TRUNK LINES CONNECTING CPU TO CONCENTRATORS
REMOTE CONCENTRATOR
MULTIPLEXERS
TERMINAL CONTROLLERS
TERMINALS

Figure 1—Network schematic

Another possible function of the Program Manager is to facilitate multiple designs and analyses under a variety of assumptions about the system and its loading. Thus, one can obtain an entire spectrum of designs for different traffic levels and evaluate the cost of providing additional throughput capacity. Similarly, the effectiveness of alternate configurations to improve reliability or response time can be evaluated.

EXAMPLES

The techniques described in the previous sections have in fact been used to design and analyze real networks. Many problems which would have been intractable to manual solution or which would have necessitated the development of costly software to solve them entirely automatically were solved by analysts using the basic design tools described above. The following illustrative examples were chosen from among these problems.

Response time analysis and central site configuration

Figure 1 is a simplified schematic of a network whose response time capability was recently analyzed using the techniques described earlier in this paper. A detailed schematic of the central processor configuration is given in Figure 2. Pairs of remote concentrators are linked to the CPU by high speed trunk lines ranging in speed from 7200 bps to 50,000

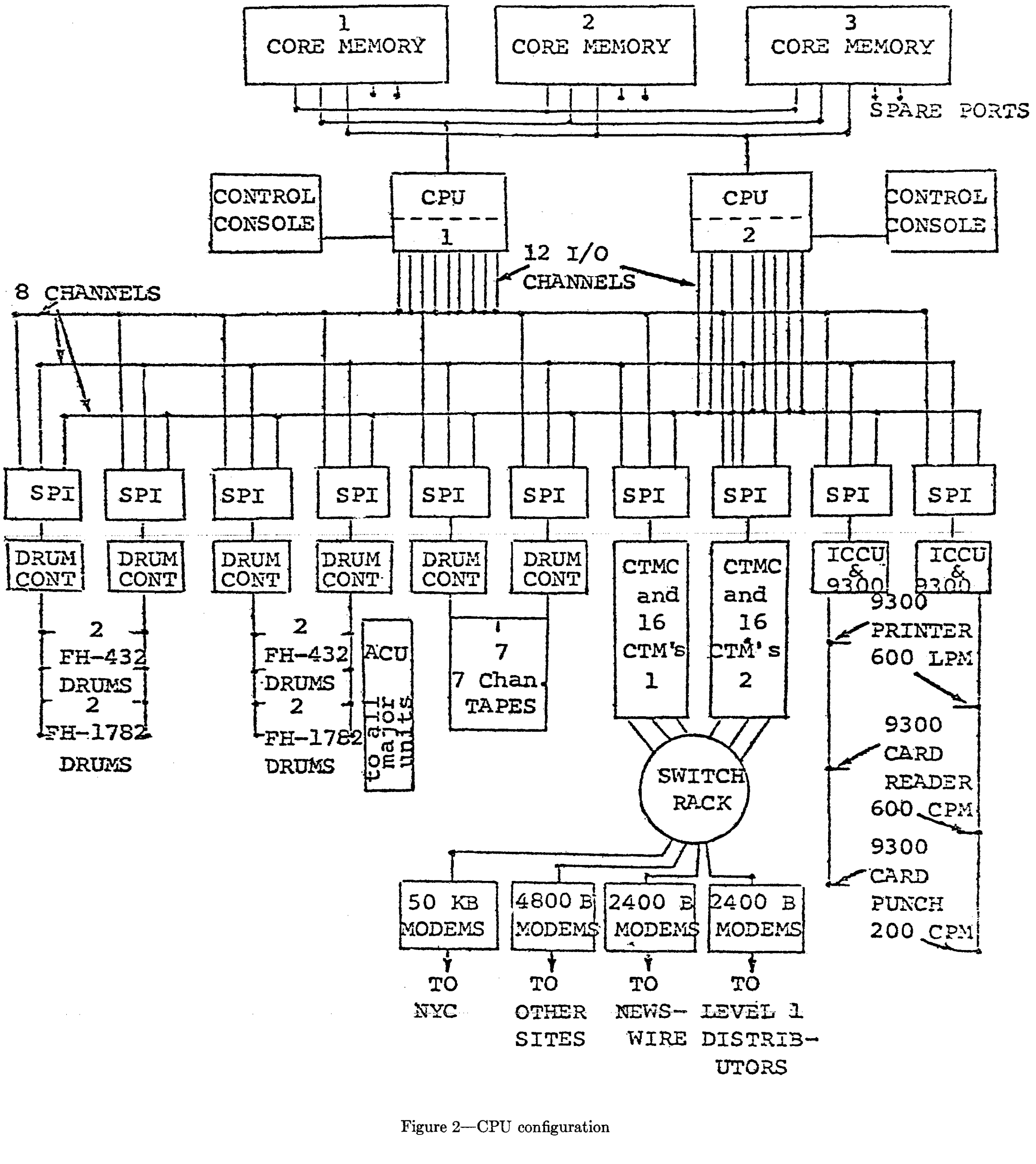
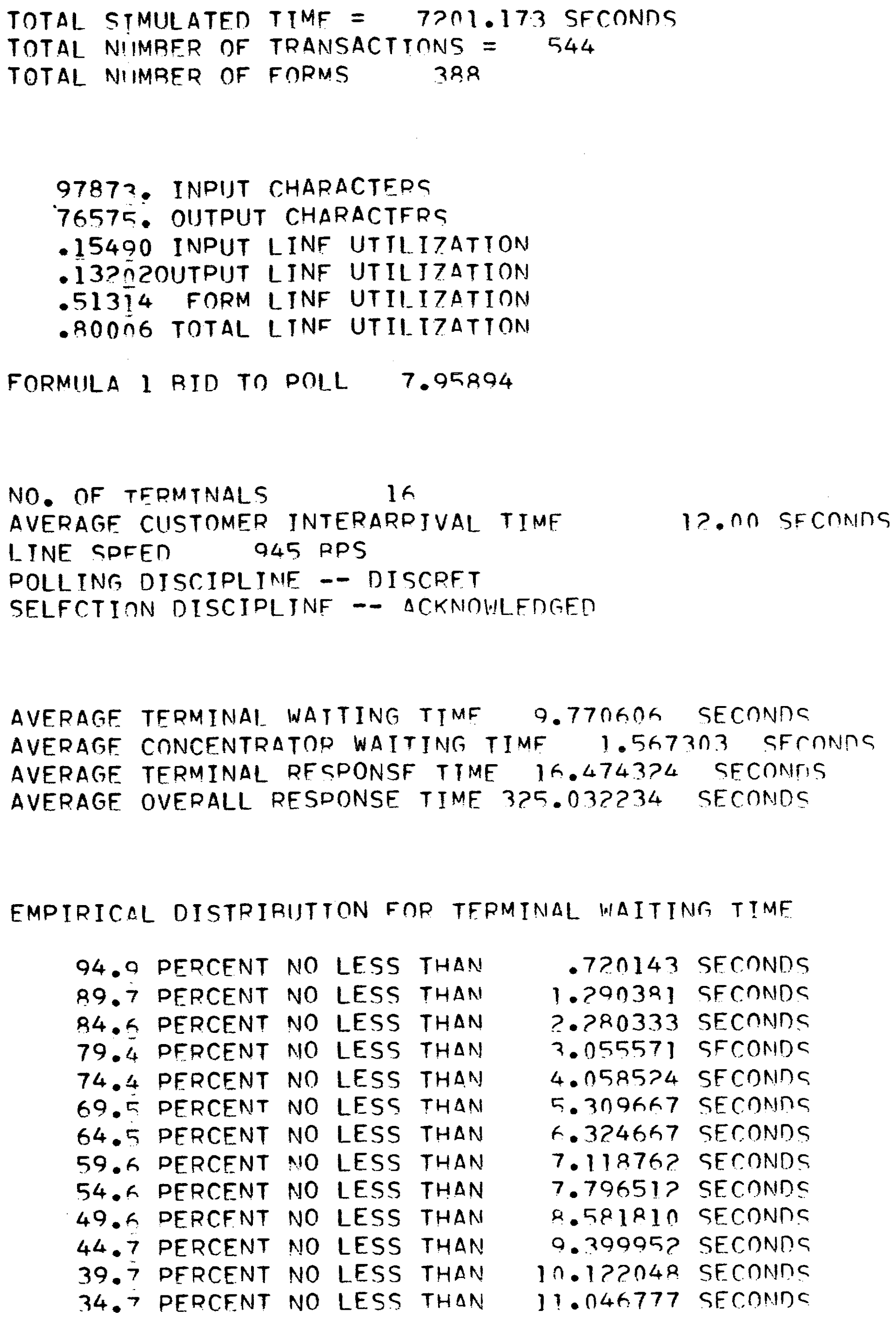


Figure 2—CPU configuration

bps. Terminal controllers are linked to the remote concentrators by multidropped full duplex voice grade lines. When economical, groups of up to five low speed lines are multiplexed onto a single high speed channel. Each terminal controller handles up to 12 terminals. The network contains over 1,000 terminal controllers and almost 2,000 terminals.

Because of the stringent response time requirements used in the design of this system, the line discipline is unusually complex. The system handles two kinds of messages with different length distributions and different arrival patterns. Polling sequences are nested into acknowledgments in order to reduce queueing delays on the regional lines. In addition, flow control procedures are imbedded in the central processor and concentrator software to prevent any possibility of buffer overflows at the concentrators and CPU. The simulation package is organized in such a way as to allow for several



```
TOTAL SIMULATED TIME =    7201.173 SECONDS
TOTAL NUMBER OF TRANSACTIONS =    544
TOTAL NUMBER OF FORMS      388


   97873. INPUT CHARACTERS
   76575. OUTPUT CHARACTERS
   .15490 INPUT LINE UTILIZATION
   .13202OUTPUT LINE UTILIZATION
   .51314  FORM LINE UTILIZATION
   .80006 TOTAL LINE UTILIZATION

FORMULA 1 BID TO POLL     7.95894


NO. OF TERMINALS         16
AVERAGE CUSTOMER INTERARRIVAL TIME            12.00 SECONDS
LINE SPEED       945 BPS
POLLING DISCIPLINE -- DISCRET
SELECTION DISCIPLINE -- ACKNOWLEDGED


AVERAGE TERMINAL WAITING TIME    9.770606   SECONDS
AVERAGE CONCENTRATOR WAITING TIME    1.567303   SECONDS
AVERAGE TERMINAL RESPONSE TIME  16.474324   SECONDS
AVERAGE OVERALL RESPONSE TIME 325.032234   SECONDS


EMPIRICAL DISTRIBUTION FOR TERMINAL WAITING TIME

   94.9 PERCENT NO LESS THAN      .720143 SECONDS
   89.7 PERCENT NO LESS THAN     1.290381 SECONDS
   84.6 PERCENT NO LESS THAN     2.280333 SECONDS
   79.4 PERCENT NO LESS THAN     3.055571 SECONDS
   74.4 PERCENT NO LESS THAN     4.058524 SECONDS
   69.5 PERCENT NO LESS THAN     5.309667 SECONDS
   64.5 PERCENT NO LESS THAN     6.324667 SECONDS
   59.6 PERCENT NO LESS THAN     7.118762 SECONDS
   54.6 PERCENT NO LESS THAN     7.796512 SECONDS
   49.6 PERCENT NO LESS THAN     8.581810 SECONDS
   44.7 PERCENT NO LESS THAN     9.399952 SECONDS
   39.7 PERCENT NO LESS THAN    10.122048 SECONDS
   34.7 PERCENT NO LESS THAN    11.046777 SECONDS
```

Figure 3—Sample output for response time analysis

distinct line disciplines and, by a change of input parameters, for a variety of polling and selection sequences for each type of line discipline. Thus, even a system as complex as the above one could be analyzed accurately.

Part of a sample output from the simulation package is shown in Figure 3. Facility utilizations, parameters describing the system, line discipline, and the length of the simulation are given first. Next, the average values of several common measures of response time performance are given. Empirical distributions of this type are useful, not only because they provide information about systemwide variation in response times, but also because they often relate directly to design constraints, such as "90 percent of the messages must have terminal response times less than 20 seconds."

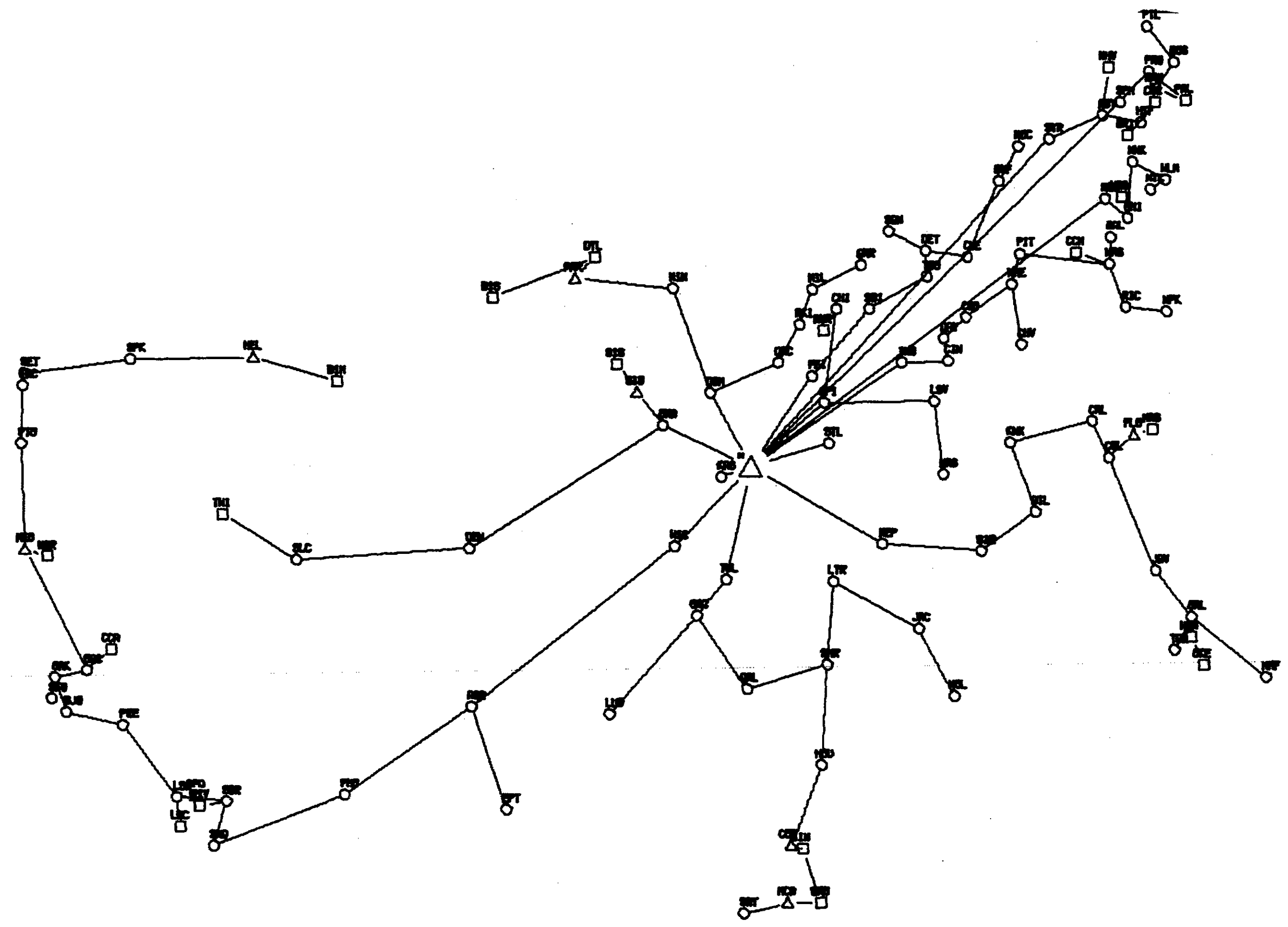


Figure 4

Terminal response time is defined as the time elapsed between the keying in of the last character of the input and the start of printing of the first character of the acknowledgment. Overall response time includes terminal response time, keying, printing, and waiting time at the terminal before the input is keyed in.

Empirical data collected from the system after it was put into operation was used to check the validity of these simulation results. The response times were found to be accurate to within five percent under normal operating conditions. Most of this variation was due to noise in the statistics. This simulation package is currently running on a CDC 6600 and requires roughly five seconds of CPU time to simulate 1,000 transactions, an adequate number for most purposes. It requires roughly 30,000 core locations to run. Because of the program's small running time and its ability to do multiple simulations on the same run by setting input parameters, it is possible to generate response times under different operating conditions very easily, thus facilitating a detailed analysis of the system's performance under a variety of assumptions about the system's loading.

On the basis of output from this simulation package, a detailed analysis of the throughput capability of the system was performed. The levels of traffic at which the central computer, the trunk lines connecting concentrators to the CPU, the concentrators, and the regional lines connecting terminal controllers to the concentrators became saturated were each identified for future planning and management of the network's operation.

Economical network design using multiplexers and concentrators

As an example of how much can be saved by incorporating concentrators and multiplexers into a telecommunications network design, a cost comparison between designs with and without concentrators and multiplexers for a network linking 102 locations across the United States is presented. The design was produced subject to the constraints that 90 percent of the transactions have response times no greater than 25 seconds during ordinary operating conditions and no greater than 35 seconds during the peak hour. Furthermore, the expected number of terminals communicating with the central

TABLE I—Network Design Summary Using Concentrators and Multiplexers

Number of Terminals	348
Number of Locations	102
Input Traffic	6,450,279 char/hr
Output Traffic	12,413,010 char/hr
Average Terminal Response Time	9.1 seconds
90th Percentile TRT	14.8 seconds
Expected Fraction of Terminals Communicating with Central Computer	.967
Tariff—AT&T 260	
Cost Summary:	
(a) Monthly Line Rental Cost	$39,263.24
(b) Monthly Concentrator and Multiplexer Cost (amortized)	25,972.60
(c) Total Monthly Cost	$65,235.84

computer in a network using multiplexers and concentrators must be no more than .1 percent lower than the expected number communicating in a comparable network without multiplexers and concentrators. A simulation package was run to determine what combinations of facility utilizations (line, concentrator, CPU, and buffer) would safely meet the response time constraints under a variety of assumptions about traffic load and mix. Designs were then produced with and without multiplexers and concentrators, using a sophisticated network design package. The design using concentrators and multiplexers is summarized in Table I.

The design using multiplexers and concentrators saved over $900,000 in annual line charges when compared with a similar design without multiplexers and concentrators. Not only was the reliability of the concentrator/multiplexer design nearly indistinguishably different from the design without them, but also, because of the concentration of key network components in a small number of areas, it was possible to provide more effective and economical backup to the system when concentrators and multiplexers were used.

As an indication of the quality of designs which can be produced by a sophisticated network design package, some of these designs were compared with proposals sent by vendors for the same network. Designs supplied by the vendors did not consider reliability, and utilized a single line utilization constraint. The comprehensive design procedure using multiplexers and concentrators had reduced line costs by over 25 percent and runs made solely for the purpose of this comparison showed savings of up to three percent over vendor supplied designs, even when identical design constraints were used.

The running time and core requirements for the design package are essentially linearly proportional to the number of locations in the system. Designs of this system required an average of 7.5 seconds of running time and 35,000 core locations on a CDC 6600. The module can be used to design networks connecting as many as 1,000 terminals.

The user can even specify the tariff to be used, subject to the restriction that it is not dependent upon the topology. Thus, even the newly proposed high/low density tariff can be used in network designs. Figure 4 shows a design obtained from this module using AT&T's proposed high/low density tariff. Not only is the design of high quality, but also, the linear dependence of core and running time on the number of locations is unaltered.

This last fact is noteworthy. The high/low density tariff charges a lower rate for lines connecting high density points, reflecting AT&T's lower costs for providing service between these points. Thus, in general, when evaluating the cost of a connection between two low density points, one must also consider the possibility of routing the line through one or more high density points, in order to take advantage of the lower tariff between these points. If done naively, this calculation would greatly increase the complexity of any algorithm for designing networks using this tariff. If proper care is taken in developing the design module, however, it is possible to incorporate the flexibility of allowing the high/low density tariff in its design procedure without sacrificing efficiency in the design process or in the design itself.

CONCLUSION

We have shown that problems of network design and analysis can be successfully solved by an analyst equipped with tools in the form of modular programs, each of which is designed for a specific purpose—input, reliability analysis, configuration, response time and throughput analysis, and network design. These tools, when used together with the analyst's experience and judgment, can produce better solutions with less effort than could be obtained by a man or machine alone.

REFERENCES

1. Bahl, L. R. and D. T. Tang, "Optimization of Concentrator Locations in Teleprocessing Networks," *Proc. of Symp. on Comp.-Comm. Networks and Tele-Traffic*, Poly. Inst. of Brooklyn, April 1972, pp. pp. 355-362.
2. Chou, W., H. Frank, and R. Van Slyke, "Simulation of Centralized Computer Communications Systems," *Data Networks Analysis and Design, 3rd Data Communications Symposium*, November 1973, pp. 121-127.
3. Frank, H., I. T. Frisch, and R. Van Slyke, "Testing the NASDAQ System—Traffic and Response Time," *Proc. of the Symp. on Comp.-Comm. Networks and Tele-Traffic*, Polytechnic Inst. of Brooklyn, April 1972, pp. 577-586.
4. Chou, W. and A. Kershenbaum, "A Unified Algorithm for Designing Multidrop Teleprocessing Networks," *Data Networks Analysis and Design, 3rd Data Communications Symposium*, November 1973, pp. 148-156.
5. Van Slyke, R., "Recursive Analysis of Network Reliability," *Networks*, Vol. 3, No. 1, 1973, pp. 81-94.
6. Whitney, V. K. M., "A Database System for the Management and Design of Telecommunications Networks," *Data Networks Analysis and Design, 3rd Data Communications Symposium*, November 1973, pp. 141-147.
7. Pan, G. S., "Communications Information System," *Telecommunications*, June 1970.
8. Doll, D. R., "Topology and Transmission Rate Considerations in the Design of Centralized Computer-Communications Networks," *IEEE Trans. on Comm. Tech.*, June 1971, pp. 339-344.

Automatic storage and retrieval system control

by P. R. WITT

IBM Corporation
Endicott, New York

INTRODUCTION

The Materials Distribution Center (MDC) at IBM Endicott, New York, is a new automated warehousing facility (Figure 1). In addition to the conventional facilities, the warehouse contains an Automatic Storage and Retrieval System (Stacker Cranes), a network of pallet conveyors, and an IBM 1800 Data Acquisition and Control System to control the Automatic Storage and Retrieval System and portions of the conveyors.* The warehouse, adjacent to the main manufacturing buildings, is for storage of raw materials, parts, and assemblies.

The computer-controlled portions of the MDC will be discussed in detail. The general background, physical layout, and material flow of the MDC are first presented to describe the environment for the computerized sections.

MDC BACKGROUND

The Endicott plant is responsible for the manufacture, assembly, and test of printers, medium-sized computers, banking equipment, and circuit boards. The manufacturing process is complex, requiring storage of the many levels of subassemblies produced in the plant. In addition to manufacturing for its own use, the Endicott plant produces parts and assemblies for use at other IBM plants and is the central supply for these items.

Prior to the construction of the new MDC, warehousing operations were fragmented and occupied 15 leased, off-site locations spread over the Endicott area. Approximately 60,000 active items were stored in the off-site locations.

The objective of creating the MDC was to bring all warehousing activities—receiving, inspection, storing, order filling, and parts shipping—on site, under one roof, and in a location adjacent to IBM Endicott's manufacturing complex.

Prior to the new Material Distribution Center, our 120,000 parts, of which 60,000 are active at any one time, were stored in over 15 off-site locations. This procedure involved a number of duplicate functions, including extensive transportation between the manufacturing plant and the off-site warehousing locations. The basic reason for building the MDC was to bring the major portion of the warehousing operation under one roof adjacent to the manufacturing buildings.

By using a central location for MDC operations a number of important objectives have been met which provided the justification for building a new warehouse facility.

The first was to substantially reduce Endicott pipeline inventory and resulted in a substantial savings to the manufacturing facility.

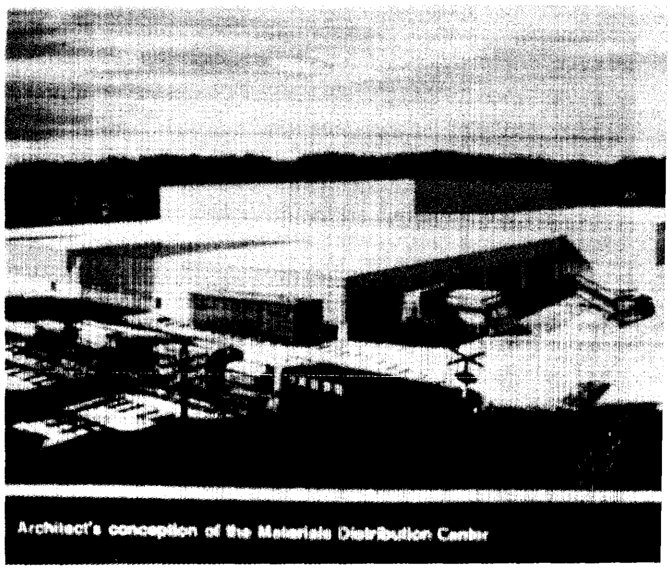


Figure 1—Materials distribution center, IBM Endicott, New York

* The following is a general description of a stacker crane system: The typical installation consists of sets of high rise storage racks, rising to heights of 100 feet or more, and extending to lengths of from 120 feet to over 800 feet, which are arranged in parallel rows in a large room. Each set of storage racks consists of two side panels containing a multiplicity of storage bins. The panels are separated by a narrow aisleway that contains a "guide rail" running along the floor of the aisleway or above the aisleway higher than the height of the highest bin. The storage-retrieval machine or "stacker" moves along the aisleway on the guide rail carrying loads to and from specific, pre-selected bins in preselected storage racks. The machine is self-propelled; it functions automatically on the basis of programmed instructions. The design of the stacker is relatively simple: it consists of a "shuttle arm" or shuttle table which extends laterally to pick up and deposit the load; an elevator carriage which raises the load to the proper height (the elevator runs along a "mast" which is a part of the stacker); and the electrical apparatus which controls both the vertical and horizontal automatic movements of the machine.

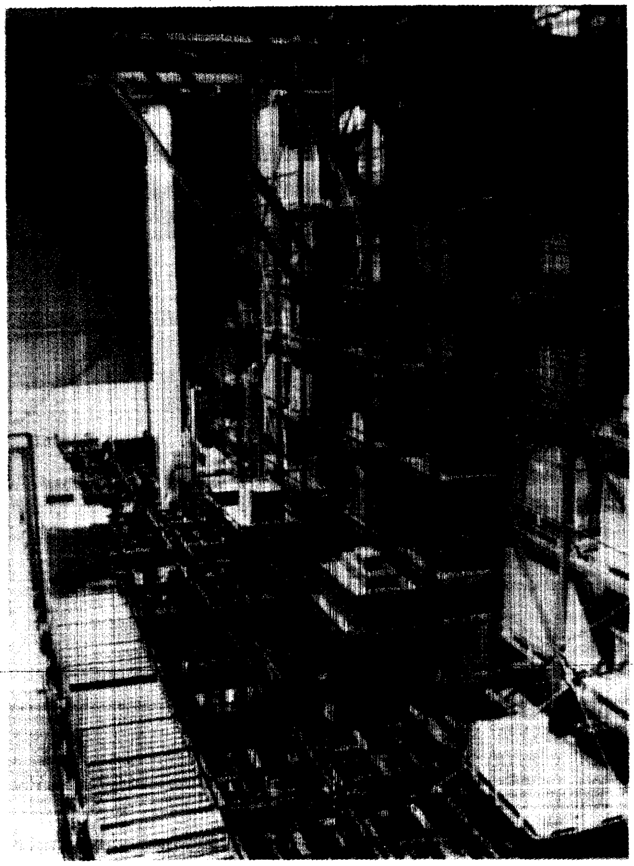


Figure 2—Automated Storage and Retrieval System (ASRS)

The second was to terminate many of the leases on the off-site locations and also have a facility that was designed for our type of warehousing.

During the design stage of the MDC, one of the problems to be solved was the fact of being landlocked in the Village of Endicott. As a solution, the use of a high-rise Automatic Storage and Retrieval System (ASRS) was investigated. A cost comparison (in terms of land, construction, equipment, and operating costs) of the ASRS with conventional truck and rack systems showed the ASRS to be a favorable choice. Because of its narrow aisles and ability to utilize high-rise storage racks, the ASRS (Figure 2) requires substantially less area than a conventional 25-foot warehouse, thereby reducing the land acquisition cost.

The ASRS also reduces manpower operating costs since it moves the product to and from the operator for storage and picking, essentially eliminating the long and unproductive transit time of men going into the storage areas to fill requisitions or store parts. The MDC, as shown in Figure 3, has 235,000 square feet in three basic portions. The first portion is a single level building (First Floor Layout, Figure 3) with 118,000 square feet of conventional warehousing. This building contains:

- the docking positions for both receiving and shipping
- an oversize bulk storage area
- bin storage for small parts
- steel and bar stock storage
- an environmental controlled room for storage of printed circuit cards
- a field service area
- parts shipping

The second area (First and Second Floor Layout, Figure 3) is a two-story building with a total of 70,000 square feet. The first floor—two-story portion—of the building contains the CBOSS area (Count, Back Order, and Sample Select). This location is for basic receiving of parts from outside vendors and other plants. Also, there is an inspection area for our quality assurance work on received materials. The second floor area contains:

- the input and picking areas for the ASRS
- the sort and accumulate area which consolidates parts into full loads for delivery to the Manufacturing building

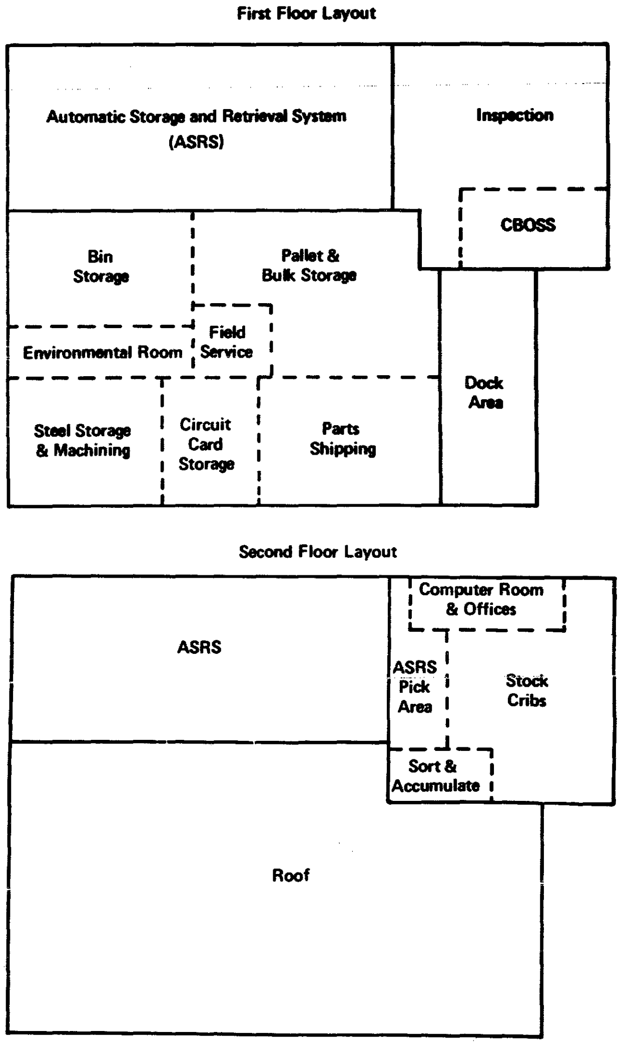


Figure 3—First and second floor layout

- cribs for special parts and tools
- the computer room and office area

The third area is the ASRS. It is single level, with a clear height of 54 feet, 47,000 square feet, and 2.5 million cubic feet of space. The area has nine 300-foot aisles with a stacker crane for each aisle. There are 18 rows of high-rise racks for storage of pallets; each crane services either side of the aisle—two rows of racks. A more detailed description is given later.

Computer simulations were used to aid in designing the Material Distribution Center. One important simulation was used in designing the ASRS. By knowing the storage and throughput requirements, a model was constructed which analyzed the various costs such as land, construction, equipment, and operating, to obtain the configuration (height, length, number of aisles) which minimized the total cost. The curve in Figure 4 indicates a minimum cost is near to the 55-foot height.

Another simulation model was used to aid in layout of the operating departments within the MDC. Using previously forecasted department space requirements and material movement volumes between departments, the model calculated the total transportation costs within the MDC for proposed layouts. This information was used to select the final layout to minimize operating costs. In addition, this data was used to determine where conveyors would be required and justified, and what throughput capacities were required.

In the ASRS, a single conveyor is used to handle both incoming and outgoing pallets to the stacker cranes. This was simulated to ensure that the single level conveyor was feasible for both control and throughput requirements. This simulation was also used to test and project the hardware system (cranes and conveyors) operation for different diverting and merging rules, partial and total sequencing of

Satisfy:	Storage volume Throughput requirement
Minimize costs:	Land Building Equipment
Compute:	1. Number of cranes 2. Amount of racks, building and land 3. Total system investment

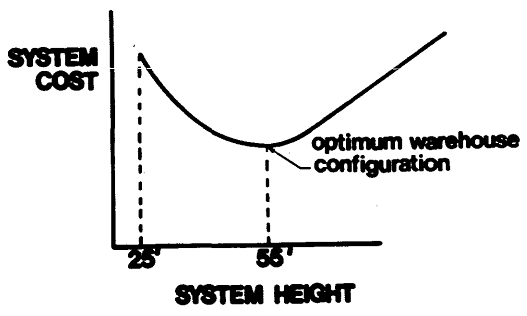


Figure 4—Graph of ASRS model results

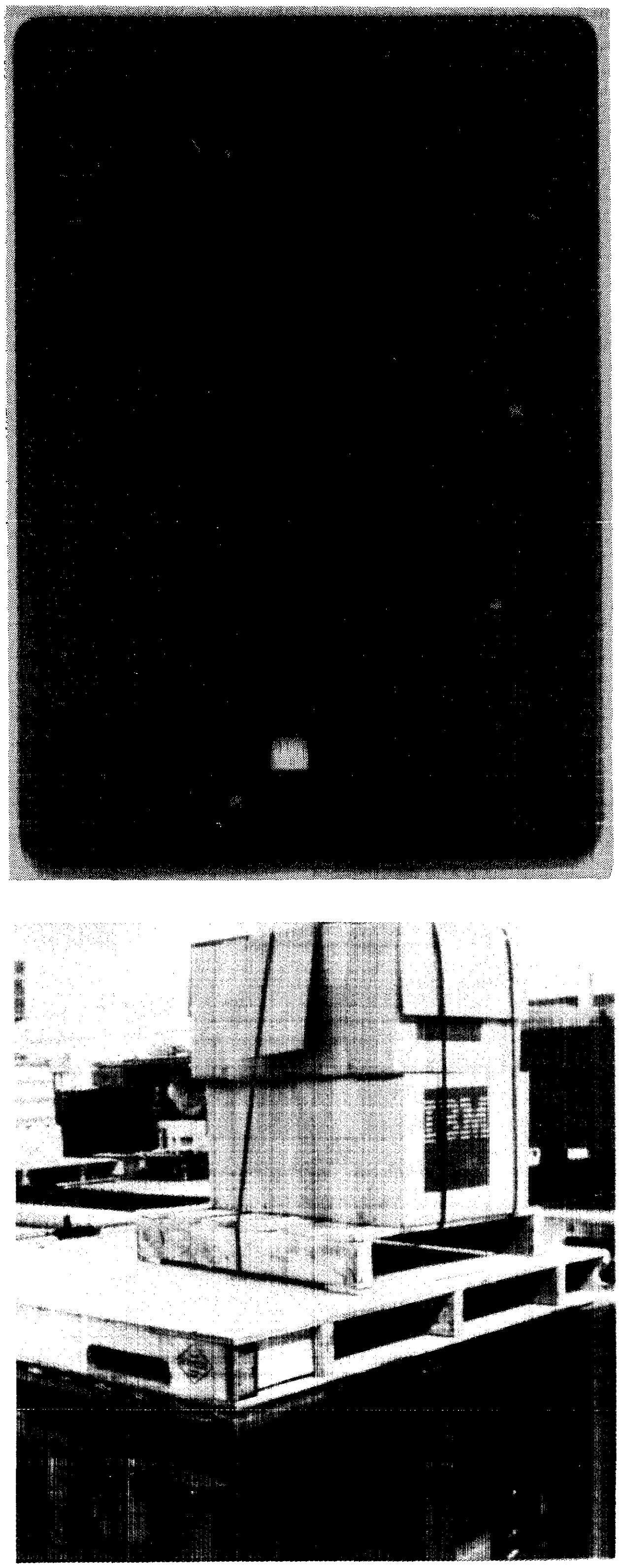


Figure 5—ASRS pallet with badge (lower center) ASRS pallet badge

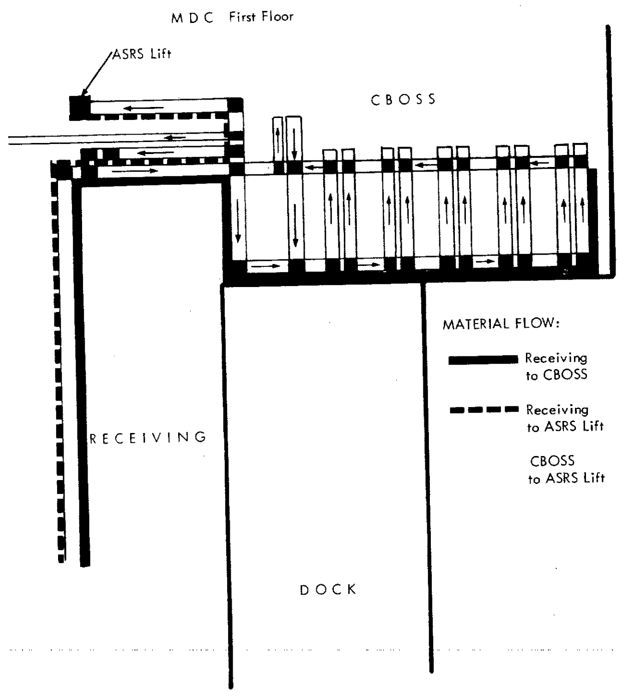


Figure 6—Material flow from receiving to ASRS

output pallets, and different command selection rules prior to actually programming of the control system. Additional programs were written for the IBM/1800 computer to simulate the cranes and conveyors to the control programs and the crane/conveyor control programs to the remainder of the control system. Also, simple programs were written to perform the actual testing of the cranes and conveyors. These programs were developed to separate the initial testing of hardware and software and portions of the application software. By this separation, the testing was simplified and shortened. Simple queuing models were used to develop the docking requirements (number docks, dock space, service times, truck waiting times).

There is material flow, via conveyor or electric truck, between most portions of the MDC. For the purpose of this paper only the flow to and from the ASRS will be outlined.

Incoming parts from vendors or other plants arrive by truck at the receiving dock. The bill of lading accompanying the shipment is used to obtain the proper receiving paperwork in the form of punched cards. These cards indicate if the item is to be sent to the ASRS, bin area, or directly to Manufacturing. The cards are placed on the parts. Parts going to the ASRS are loaded onto the special ASRS captive pallets. These pallets are 40 inches by 52 inches or 40 inches by 62 inches. Each pallet has two badges, which are on diagonal corners (Figure 5).

Each badge has a reflective and non-reflective side to be used for control of the pallet routings through the conveyor network on the first floor. Also, there is a number punched in the badge that is to control the flow through the ASRS conveyor network on the second floor.

The man at the receiving area orients the two badges for the desired routing. As the pallet flows through the network of conveyors, the elective photo-cells check for reflection or non-reflection of the two badges. This information controls the routing through the various portions of the downstairs conveyor.

The flow of ASRS parts from Receiving to the ASRS is shown in Figure 6. Pallets can move directly to the ASRS lift from the dock (dashed line 1). However, most ASRS parts are initially sent through the CBOSS (Count, Back Order, Sample Select) area. In the CBOSS the pallets are directed to one of ten identical spurs.

Here the basic receiving of goods is completed. The parts are counted and the receiving paperwork completed. Back orders are filled. If the parts require inspection, a sample lot is selected and sent to the Quality Department. ASRS parts are then released on the main line conveyor at the end of the CBOSS spurs and directed to a lift which takes them to the second floor. The pallets enter the lift and are brought to the input station for the ASRS (Figure 7). Here data is entered into the terminal describing the parts and pallets entering the system.

The individual pallets flow through a physical-sized sensing for proper matching to the appropriate slot size in the ASRS. Then each pallet moves on the conveyor to the selected aisle and is stored by the cranes.

As parts are required for manufacturing, requests are sent to the warehouse. In the ASRS, parts are retrieved by the stacker cranes and placed on the conveyor which delivers the pallets to one of the seven identical picking spurs in the pick area (Figure 7). The parts are picked and the activity reported through a terminal. Parts that remain on the pallet after picking are recycled back into the stacker crane system for restorage. The parts that have been picked move along the conveyor to a Sort and Accumulate area.

In the Sort and Accumulate, the parts are sorted by location in the manufacturing buildings and consolidated into full loads for delivery to manufacturing. From the sort and accumulate area, pallets move on a conveyor to the manu-

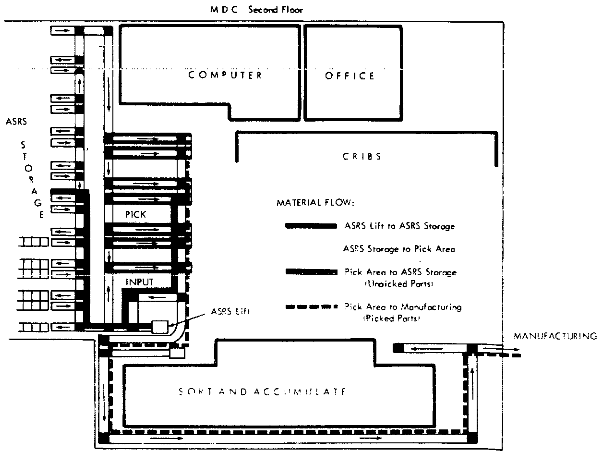

M D C Second Floor
COMPUTER
OFFICE
ASRS
STORAGE
CRIBS
PICK
INPUT
ASRS Lift
MATERIAL FLOW:
ASRS Lift to ASRS Storage
ASRS Storage to Pick Area
Pick Area to ASRS Storage (Unpicked Parts)
Pick Area to Manufacturing (Picked Parts)
MANUFACTURING
SORT AND ACCUMULATE

Figure 7—Material flow ASRS to manufacturing

facturing facility. This description briefly summarizes the material movement to and from the ASRS within the Material Distribution Center.

COMPUTER CONTROL—PHYSICAL AREA

The portions under computer control are the ASRS (stacker cranes) and the conveyors on the second floor that service the stacker cranes for both input and output of pallets. These are shown in Figure 8 and consist of four basic areas:

Input area

- As shown in Figure 8, new Pallet Spur (A) enters pallets coming from the receiving dock or CBOSS via the lift from the first floor into the system.
- Recycle Spur (B) returns pallets that have been retrieved and partially picked to the system with the remaining parts.
- Size Sense Station (C) determines pallet load sizes.
- Reject Spur (D) handles pallets failing to meet input criteria—either sizing or data. These pallets are tracked onto Pick Spur S7 for correction of deficiencies and reentry into the system.

Pick area

- Pick Spurs (S1-S7) handle the retrieved pallets for picking and counting. Each spur has space for seven pallets. An eighth position (SS) on the spur is for 40-inch by 48-inch slip sheets used for taking away parts that have been picked. There is a computer-controlled physical gate between this position and the first pallet on these spurs. This allows for release of pallets from the pick spurs under the proper conditions.
- Take-away Line (V) returns partially picked pallets to the system via Spur B and carries parts to Sort and Accumulate.

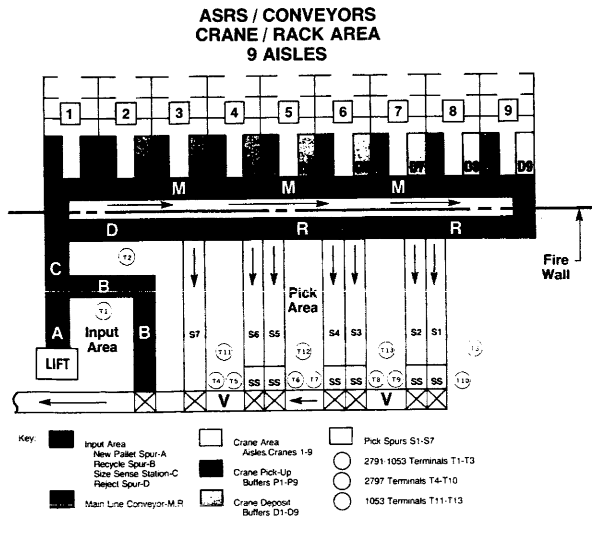


Figure 8—Schematic of areas under computer control

Conveyor Area

- As shown in Figure 8, Mainline Conveyor (M) handles pallets going to and coming from the cranes. It carries input and output pallets simultaneously.
- Pick Area Mainline Conveyor (R) carries the pallets to the pick spurs.
- Input Buffers (P1-P9) are conveyor spurs, one for each aisle/crane, that handles pallets going into a crane/aisle. Each buffer has capacity for three pallets going to the aisle. The position furthest from the Mainline Conveyor (M) is the pickup position for a pallet to be stored by the crane. The mainline and input/output buffers are level with the vertical midpoint of the rack structure at the front of the ASRS.
- Output Buffers (D1-D9) are conveyor spurs, one for each aisle/crane, that handle pallets coming out of an aisle/crane. Each buffer has capacity for three pallets. The position furthest from the Mainline Conveyor (M) is the deposit position for a pallet retrieved by the crane. The conveyors operate at a rate of 45 feet per minute.

Stacker crane area

- Stacker Cranes (1-9) transport (store and retrieve) the pallets within the storage-rack structure. There is one crane for each aisle. The crane can be communicated with anywhere in the aisle upon completion of the previous command; there is no home base. The crane travels on rails at a maximum horizontal rate of 300 feet per minute. The shuttle, which carries the pallet, travels vertically at 45 feet per minute.
- The rack structure contains the slots for storage of the pallets. There are 18 rows of racks, with each crane servicing two rows: one row to the right and one row to the left of the crane aisle. Each row is 78 bays long. The racks on Aisles 1 through 7 have 12 tiers. They accommodate pallets 52 inches long by 40 inches wide. Slot heights are 30 inches, 43 inches, and 79 inches, with each horizontal level having the same height. The racks on Aisles 8 and 9 have eight tiers accommodating pallets 62 inches long by 40 inches wide. Slot heights are 62 inches and 89 inches. The weight limitation is 2000 pounds per pallet.

COMPUTER CONTROL—GENERAL

The basic functions of the computer-control system are to direct the conveyor-flow, storage, and retrieval of pallets and to maintain a status of the pallets in terms of contents and location. All controlling and data keeping are done relative to unique pallet numbers.

The first functions of directing pallet flow, storage, and retrieval involve the monitoring and controlling of conveyors and cranes by the computer. This controlling of physical devices requires communication between the computer and the physical devices.* There are approximately 350 sensors

* All hardware/computer communication is in the form of Process Interrupt, Digital Input, and Digital Output. These can be viewed as signals containing one bit of information (on/off, yes/no).

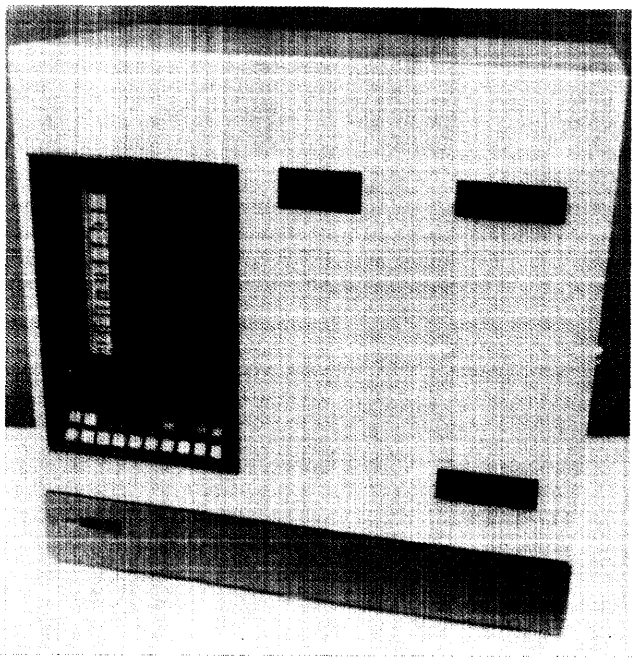


Figure 9—IBM 2791 area station

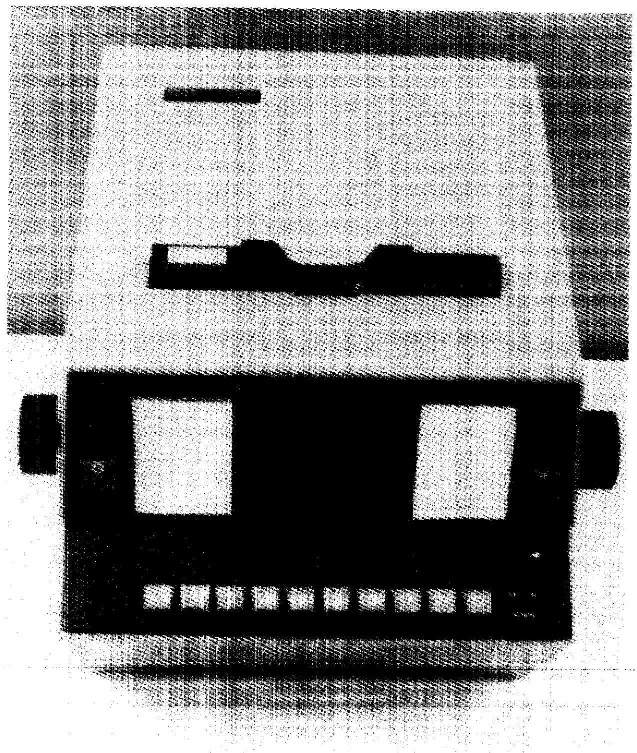


Figure 10—IBM 2797 data entry unit

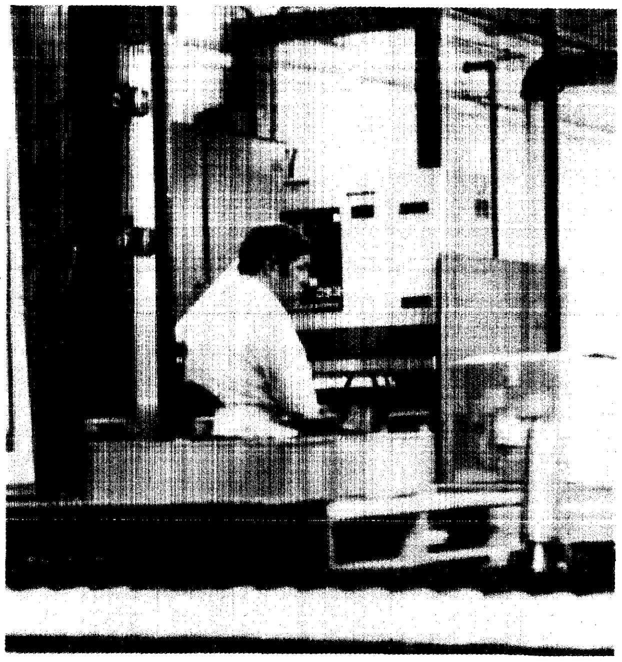


Figure 11—Input operator entering data into the 2791

(switches and photocells) throughout the ASRS and conveyor network. When a pallet activates a sensor (by breaking a light beam, for example), a signal is sent to the computer. The computer analyzes the signal and determines what event has occurred. When the computer wants a device to perform a task, it sends out a signal that causes a physical action, such as raising a pallet stop or causing the pallet to divert onto another conveyor spur.

Signals representing commands are sent to the cranes, the logic circuitry of which interpret and perform the commands. In turn, the cranes send signals to the computer to report what has been accomplished. Simple signals form the communication means of monitoring and controlling the physical devices.

Maintaining the status of pallets in the system requires data files that identify the parts on a pallet and document where they are stored and the quantities on each pallet. Activity affecting the status (picking of parts, change of location, or entering of new parts) must be captured to update the status. To use the cranes and conveyors to meet the varied warehousing tasks effectively in the dynamic ASRS, current data must be continually available. To do this, the files are online to the computer on disk drives.

The updating of the disk files is done online in real-time through the use of terminals. IBM 2791, 2797, and 1053 terminals are used to communicate data between the workers and the computer. The 2791 (Figure 9) accepts badge, punched-card, and keyed-in data. The 2797 (Figure 10) accepts badge and keyed-in data. The 2791s and 2797s are used by the people in the input and pick areas to report activity on the pallets. The 1053s are typewriter terminals used to print messages that provide guidance in reporting activities and identifying errors. The placement of the 14 terminals is shown in Figure 8. Online files and update not only substantially improve the physical operation but also improve the integrity of data by immediate reporting and auditing of activity. The need for data integrity is much greater in a stacker-crane system than in conventional warehousing since it is impossible to make an easy visual scan of the storage area.

An IBM 1800 Data Acquisition and Control System provides control and information; it communicates with the physical devices, the terminals, and the files.

Use of a realtime information system to dynamically drive the system controlling the cranes and conveyors is the key to making the ASRS meet the varying warehouse needs. The tendency has been to limit warehouse flexibility in order to simplify the control system. It should be noted that many of the features of the system that will be described do not bear directly on the relatively simple problem of sending commands to the cranes and conveyors. Instead, these features are aimed at dynamic development of control decisions, to increase the productivity of warehouse operating and management personnel.

SYSTEM OPERATION—INPUT

Pallet selection

Pallets arrive for input to the ASRS on the New Pallet Spur (A, Figure 8) via the lift from the first floor and on the Recycle Spur (B). The operator stationed at the 2791 Terminal T1 selects the next pallet to be processed from either Spur A or Spur B.

Pallet identification

The operator removes the badge from the pallet and inserts it into the 2791 (Figure 11). This identifies the pallet being processed.

Part identification

For pallets entering the system with a load not previously stored (normally from Spur A), the operator removes an 80-column punched card or cards from the pallet. The cards identify the parts on the pallet by part number. Two part numbers are allowed on new, non-inspect shipments. Documents accompanying the cards indicate the quantities of each part and the type of load. These documents are in the form of a two-part card. One part stays at the input station for keying in data and for later auditing of the input transaction after the working shift. The second portion remains with the pallet for identification of the parts during picking.

Transaction selection

The operator then enters the type of transaction. This indicates the data to be expected, the audits to be performed on the data, how the data are to be processed, and which files are to be updated.

- New receipt: This pallet contains parts of one or two part numbers to be stored in the system and is available for retrieval as soon as desired.
- Parent: The pallet is to be stored in the system but is to be "bin-locked" (made unavailable for retrieval) until the sample has passed inspection and shipment has been accepted. The bin-lock of the pallet is done under the part number and purchase order number, found in the punched card identifying the part, and the keyed-in shipment number.
- Sample: This pallet contains a sample that has been accepted. It is to be stored in the system, and any pallets previously locked under that same purchase-order number and shipment number are to be unlocked.
- Recycle: This pallet is returning from the pick area for reentry into the system. Because the files have been previously updated, no other data except from the badge is needed for this transaction.
- Reject: The operator wants to reject the pallet for some noticeable error condition, such as absence of paperwork or a load that appears unstable. He also keys in a reason code for the rejection.

Data entry

For New Receipt, Parent, or Sample, the operator enters the punched card(s) (receiving document) into the 2791. The part number and qualifiers are read from the card; for samples and parents, the purchase order number is also read. The operator keys in the part-number quantity on the pallet. For parents and samples, the shipment number is also keyed in.

Data audit

At this point the system knows what pallet, what parts, what quantity, and how to process the data. The system does a preliminary audit on the transaction, posing queries such as:

Is this a valid ASRS part number?
Is it a valid recycle?
Has all data been entered?
Do data fit the limits?

If the transaction fails, the system may request a repeat of the transaction or may set up a rejection of the pallet.

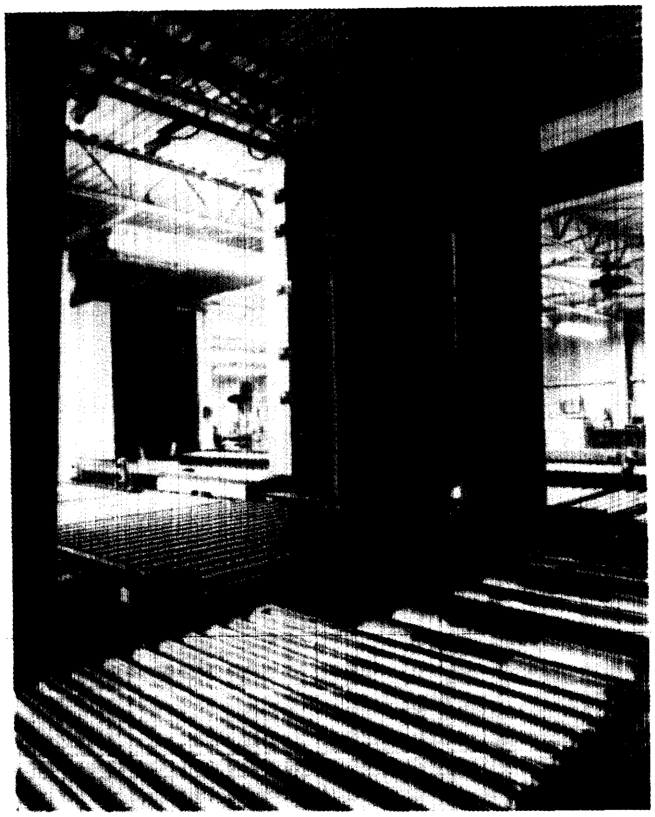


Figure 12—Size sense station

Transaction completion

At the completion of the transaction, the operator replaces the badge and presses a button next to the spur holding the pallet (either A or B). This signals the system that the operator is physically finished with the pallet and indicates which spur the pallet is on.

Pallet movement

When the transfer leading to Size Sense Station (C) is empty, the computer signals either Conveyor A or B to move a pallet. This pallet will move from the transfer area as soon as there is no pallet in the size sense station.

Size sensing

In the size sense station (Figure 12), the physical characteristics of the load are determined. By analyzing the photocell beams that are broken by the load, the computer determines the length of the pallet (52 inches or 62 inches) and the height (including over-height). Signals detecting load overhang on any side of the pallet or overweight (over 2000 pounds) are sent to the computer. The system now knows the physical size of the load and any out-of-limit conditions.

Pallet rejection

Entry of the pallet will not be allowed into the system for storage if a physical characteristic is out of limits, or if the transaction failed the audit, or if the operator chose to reject. The computer then causes the pallet to be diverted onto the Reject Spur, (D), and onto the Pick Spur (S7), and a message is printed by the 1053 Terminal servicing S7. This reports the pallet number rejected and the cause for rejection. The pallet is then either removed from the system or the problem corrected and entered again via Recycle Spur (B).

Aisle selection

If the pallet is accepted for storage, the location is dynamically selected from all available slots. There is no permanent tie between the physical slots and either pallets or part numbers. Each time a pallet enters the system, a new location which is best for system operation at this time is chosen (recycles will probably not return to their previous slots). To select the location, a number of steps are performed to progressively eliminate aisles from consideration. Some of the steps are:

- For 52-inch-long pallets, Aisles 1 through 7 are considered; 62-inch-long pallets can only be stored in Aisles 8 and 9.
- If a crane is offline, it is eliminated from consideration.
- One of the three input-buffer positions must be free.
- The percentage of proper-height slots still available in the various aisles is also evaluated against certain limits.
- If another pallet holding the same part number is already stored in the system, an attempt is made to put the new pallet into a different aisle. This is done to reduce the risk of parts being unavailable because of crane downtime.
- If more than one aisle satisfies the conditions, a cyclic selection of an aisle is made.

At this point, an aisle has been selected with at least one slot of the required size free. The actual slot will be dynamically selected at the time of storage.

File update

The data files are updated from the information in the transaction.* For newly entering pallets:

- The pallet file is updated with the part numbers, quantities, and status, such as bin-lock and in-transit.
- The pallet is added to the chain of pallets already in the ASRS under the part number only, or under the part number, purchase order, and shipment number (for inspect-locked items).

* See Appendix for details of files.

- The total of pallet quantities in the part-number file is incremented, and indicators are changed to reflect the addition of a new pallet.
- Parent file indicators are changed to reflect the addition of a new inspection bin-locked pallet.

Transaction records

Throughout the various areas whenever a file is modified or an error or potential error is encountered, a magnetic tape record is written. This allows files to be rebuilt if they are destroyed during live operation. The tape is also used to produce batch reports, particularly for auditing.

SYSTEM OPERATION—PICK AREA

Pick-area tasks

Two major functions are performed in the pick area:

- Picking of parts (requisition filling)
- Counting (rotating inventory counts—RIC)

RIC is done in place of a once-a-year total physical inventory in the warehouse. Parts are periodically counted, depending on their dollar value and activity, so that the physical counting is spread throughout the year.

Two minor functions are performed in the pick area:

- Auditing of errors or potential errors found in the actual operation of the stacker-crane system.
- Retrieval of inspect bin-lock items for reinspection by Quality.

Requisition and count requests

Two groups of punched cards are received daily from other systems in the plant:

- Requisitions. Each requisition is for one specific part number and quantity. There can be several requisitions with the same part number but with different departments; these will be on separate requisitions.
- Requests for counts. Each card contains a part number that is to be counted in its entirety. The counts will later be compared against the book record contained in other inventory systems.

Batches

The two groups of cards are kept separate. The requisitions are sorted by part number so that all requisitions for a given part number fall together and will be filled at the same time to minimize pallet traffic. Each group is then separated into small batches, each batch containing what is estimated to be one to two hours' worth of work at a pick station.

By the use of these batches, the pick area management is able to spread the work and balance the work load across pick spurs. These batches will be assigned to various spurs throughout the day. This is done rather than building up total work for a given spur before starting, and thus possibly resulting in one spur with ten hours' worth of work and a second spur with six hours' worth of work.

Preparation for live operation

Prior to starting daily operations, the batches of requisitions and count cards are processed by the 1800 against the pallet and part-inventory files. Crane commands for retrieval are *not* formatted at this point.

For each batch, a small disk file is built. The disk file for each batch contains the same part numbers as found in the cards within the batch. However, the disk-file records are summary records. For instance, if there were five requisitions calling for Part A, each having a quantity of 10 within a given batch, the disk-file record for Part A would occur only once with a total quantity of 50 needed to fill all requisitions.

For RIC batches, the disk file contains the part numbers and the number of the pallets to be counted for each part number. Pallets to be counted are RIC bin-locked.

These disk files will be used during live operation to determine what pallets must be brought out to fill the requisitions or to satisfy the request for counting. The pallets will be determined dynamically at the time the work is to be done, and the crane commands will be generated at that point.

As the batches are processed against the files, any requisitions that do not have available parts in the system are separated out for the back-order file.

A report for the pick-area technician is produced showing:

- For each batch the pallets that are anticipated for retrieval. They may or may not be the pallets that come out as a result of the changing status of the system during operation.
- Requisitions that may be only partially filled because of lack of parts in the system.
- Conflicts between counting and picking. From this, the technician will attempt to do the counting batch early in the day and the picking late in the day, so that the same part number can be both counted and picked in one day.
- A summary of anticipated crane usage for each batch. The technician uses this data to keep from assigning a number of batches that use one or two cranes heavily, thereby slowing down the system.

The report is used to guide the technician in assigning and balancing work over the seven pick spurs. The data is not used by the pickers or counters in performing their work. They use the actual request-for-count and requisition cards in doing the work.

Batch assignment

The system is now ready for live operation. The pick technician assigns batches of work to the various spurs as follows:

- He hands Batch 20 (a pack of requisitions or count requests) to the worker on Spur #4.
- Through a 2791 terminal, he tells the computer that the punched cards for Batch 20 are a Pick Spur #4.
- The computer searches the disk file for Batch 20 and knows the part numbers and quantities needed to fill the requisitions at spur #4. This information will be used to select and retrieve pallets to satisfy the requisitions.
- The technician may suspend a batch and reassign it to another pick spur in order to balance the work load.
- He may also assign a batch in a pending mode. In this way, when the current batch is finished, the computer will automatically begin allocating work from the pending batch.

By using the above methods, he is able to balance the work and have various numbers of spurs working at different times during the day.

General pick-area operation

The operations performed in the pick area consist of filling requisitions and counting parts.

The operator in a spur works off the first and only the first pallet in the spur (S1-S7, Figure 8). He then reports the activity on the first pallet. If he reports on any pallet other than the first one, the transaction will not be accepted by the system. Until he has completed transactions associated with a pallet and it is physically removed from the spur, he cannot report on the next pallet. This is done to maintain integrity and to smooth the flow in the system. Tremendous peaks and valleys could be created if reporting were to be done on a number of pallets within a spur and then released in a short period of time.

When the man has finished on the pallet, he reports on the 2797 terminal that he has completed pallet processing. At this point, the files are updated with the data he has reported. Once this is done, a signal is sent to drop a physical gate that separates spur position 1 and slip sheet position; this allows the man to remove the pallet. Until he has reported completely on a pallet, he is not able to physically remove the pallet. This substantially aids the integrity of reporting on work completed.

When the gate is dropped, the man physically pushes the pallet out of the spur and onto conveyor segment V (Figure 8) for delivery to either sort and accumulate (if the full load has been picked), or to return the unpicked portion of a pallet to the system via segment B.

As the pallet is pushed out of Position 1, a signal is sent back to the computer indicating there is now a position free on the spur for another pallet. This signal is essentially a request for the computer to select and retrieve another pallet. Until the first pallet has been physically removed, the new pallet will not be allocated. This is done so that if the man leaves the spur for any reason after finishing reporting on a pallet, another pallet will not be sent to a full spur, causing a jam on the conveyor.

Full pick spur

In each spur, there are seven positions for pallets. When the request for another pallet is initiated, it is really for a pallet seven ahead of where the man is now working. The system attempts to keep each spur as full as possible to smooth the variation in picking time over different pallets. To do this, seven pallets will be allocated to a spur, as long as work is assigned to that spur. The seven will be:

- Physically in the spur
- On Mainline Conveyors R or M leading to the spur (Figure 8)
- In course of being serviced by the crane
- Waiting to be serviced by the crane

Pallet selection

When the signal indicating the release of the pallet is received, the computer goes through the process to allocate another pallet. It relates the signal from a specific spur to the current batch assigned to that spur. Through the disk file for each batch, it determines where in the file it last allocated a pallet. It looks at the summary record of the part number for the previously allocated pallet to determine whether that part-number request has been satisfied. If not, it will look for another pallet with that part number in the system and request retrieval for it. Each time a request for another pallet comes in, it will continue to look for another pallet under that part number until:

- The total quantity for requisitions has been satisfied.
- All the pallets needed for a valid RIC have been retrieved.
- None are left in the system.

If the previous part number request has been completed, the system will look to see if any emergency requests are pending for the spur. If no emergencies are pending, the system goes on to the next part number in the disk file for the batch.

If this was the last part number in the batch, it will automatically go on to the first part number in the next batch assigned to the spur. Once the next pallet has been selected, the appropriate data are passed to the crane control programs, which will set up the appropriate commands to retrieve the pallet.

It should be noted that even if multiple pallets are required to satisfy the total quantity needed, the pallets are selected and requested one at a time as a space becomes available on the spur.

Smallest quantities first

Pallets retrieved for picking are selected in the sequence of smallest quantity first. For instance, if a total quantity of 100 was needed to fill all requisitions for the part number and there were two pallets in the system, one with 50 and a second with 150 parts, the pallet with 50 would be requested first and the pallet with 150 would be requested second. This is done to empty pallets and conserve space. To call out only the pallet with 150 parts would cause two partially full pallets to be in the system. This pulling of pallets containing the smallest quantity first also eliminates the need to marry parents and samples that have been stored at different times. The sample, normally occupying a small portion of the pallet, is stored later than the parent. However, because of pulling by smallest quantity first, it will be retrieved first and emptied. Thus for a short period of time some storage utilization is lost, but in return, the unproductive and costly work of retrieving the parent to marry with the sample is eliminated. If multiple pallets are retrieved, all will empty with the possible exception of the last.

Partial sequencing

When more than one pallet is being retrieved to satisfy the requisitions for a part number, sequencing is needed. If both the pallet with 50 and the pallet with 150 are called out, the pallet with 50 must arrive first, or the reasons for bringing out both will be defeated. Under these conditions, the pallets must arrive in the sequence they were requested. When single pallets are requested for parts, it is immaterial whether the first pallet requested arrives before the second pallet requested. Sequencing is also done on RIC pallets to make sure that all pallets for a part number arrive together.

Hence, the system performs partial sequencing. The use of partial sequencing, rather than total sequencing, substantially reduces the loss of effective capacity throughout the system. For instance, 50 percent sequencing has only 25 percent of the impact of full sequencing in loss of flow through the conveyor network. This sequencing data is also passed along with the crane commands for use in the conveyor tracking.

Advantages of dynamic retrieval selection

The pallets and the commands are selected dynamically as the parts are worked on, rather than preformatted at the beginning of the day. This provides for better operation in the system; for example:

- Suppose the crane on the aisle containing the pallet that would be first selected is down (out-of-service). If the commands were preformatted, they would be bypassed. By dynamically selecting, the system automatically looks for any other pallets that are in the system and withdraws any required to satisfy the requisitions.
- Requisitions may be scheduled to be only partially filled because of the number of parts and pallets in the system at the beginning of the day. If new pallets are received and/or released from inspection bin-lock, they will be dynamically allocated, eliminating the partially filled requisitions.
- RIC-locked pallets may be picked from, provided the count is done before picking.
- The effect of emergency requisitions on normal picking can be minimized. If an emergency has been filled from a pallet early in the day, the pallet may no longer have enough parts to fill the planned requisitions. If the commands were preformatted, the planned requisitions would be only partially filled. However, with dynamic selection, the system picks up this condition and allocates additional pallets to fill all the requisitions.

Reporting in the pick area

As an operator works on a pallet, he reports the activity he has completed to the system. He does this by placing the pallet badge in the 2797 terminal and entering type of transaction he has performed. He also fills out the requisition card or count card with the correct data.

For requisition filling the sequence is as follows:

- As an operator fills each requisition, he enters the quantity pulled in the 2797 (Figure 13). A running total of the quantities reported on a pallet is accumulated.
- When he has finished processing the pallet, the operator may find one of two conditions exists: (1) he has unsatisfied requisitions with no corresponding parts on the pallet (the pallet may or may not have another type of part on it) or (2) he has filled the requisitions but there are extra parts of the same kind on the pallet. In the first case he enters a "zero part" transaction. This indicates to the system that he is finished with the pallet and there are no more parts. If the system records show there should still be parts on the pallet, the transaction is questioned by a message on the 1053 Printer. If the operator repeats the "zero parts" transaction, it is accepted. In the second case, he enters a "finished-pallet" transaction. If the system records show that it should be zero, the transaction is questioned. If he reenters a finished-pallet transaction, a dummy quantity of one (1) is placed in the pallet record and the pallet marked for audit by warehouse personnel.
- At the end of a reporting on a pallet, quantities reported are used to update the pallet and part-number inventories.
- If the operator has unfilled requisitions for a part number, but the next pallet does not contain that part number, he knows they will remain unfilled since all

Figure 13—Pick area operation and reporting

pallets with the same part number would have come out together. He puts these aside and works on the requisitions that match the part number on the pallet.

For RIC counting the sequence is as follows:

- The operator enters the pallet badge and the total count for the pallet.
- This is accepted by the system, and file quantities are updated.
- During counting, pallets can be merged by zero-counting one pallet, and incrementing the count on a second pallet containing the merged parts.
- If one or more of the pallets required for a complete count are unavailable, the system cancels the count activity for the part number via a message to the counter.

The operators also enter two other transactions. The first is a "no action" transaction: the operator recognizes the part number but he is not going to do any work on it. For instance, it is the end of the day, and there are one or two pallets he wants to remove from the spur so that it is empty. The second is an "unexpected" transaction. This essentially means that he does not have requisitions or count cards to cover the part(s) on this pallet. This may or may not be recognizable by the system. The pallet may have been erroneously diverted into the spur because of hardware failure. The system knows this and knows the operator has no matching requisition. In another case, the system recognizes the pallet number as being legitimate but the pallet obviously does not contain the part recorded in the file; this is marked for audit.

When a transaction is incorrect or questioned by the system, a light is turned on next to the 2797 terminal to get the worker's attention. A message is printed on the 1053 and the worker takes the proper action.

The pick area technician has a number of 2791 transactions by which he can control the operation in the system and override or add to the planned retrieval of pallets.

- Batch Assign/Suspend. This has been previously discussed.
- Emergencies. He can enter requests to retrieve pallets to fill emergency requisitions, assigning these to any spur that is working on requisitions. The requests for emergency pallets take priority over the planned picking.
- Management Bin-Lock/Unlock. This allows locking or unlocking under 15 separate codes of specific pallets or all pallets under a part number by management directive.
- Inspect. This retrieves inspection bin-locked pallets, which cannot be normally withdrawn, for return to vendor or resampling by quality personnel.
- Retrieve by Part or Pallet. A single pallet or all pallets

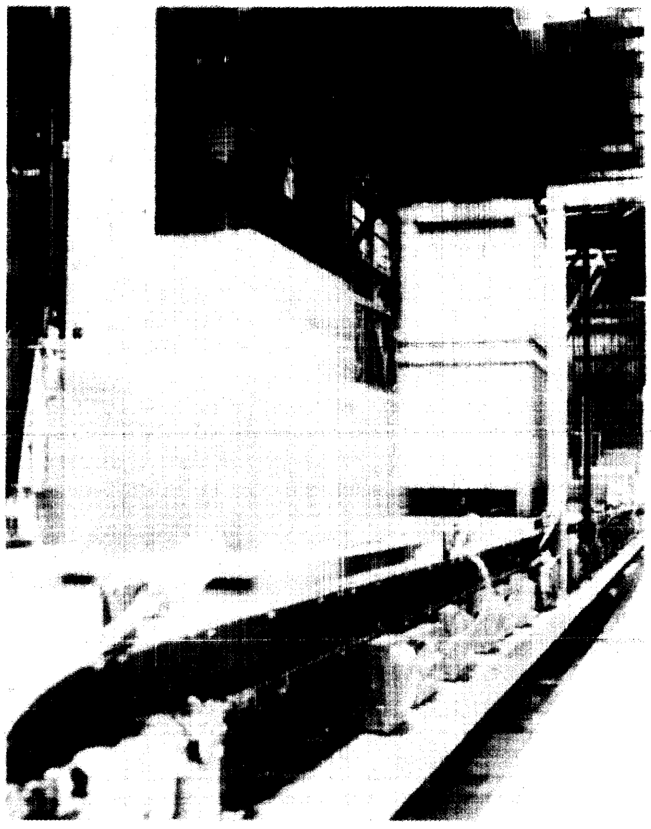


Figure 14—Pallets moving on main line conveyor

containing a part number can be retrieved. This is used to perform auditing of problems or potential problems trapped by the system.

- Inquiry. Inquiry can be made to any of the files on the system.

SYSTEM OPERATION—CONVEYOR CONTROL

Simultaneous input/output flow

The Mainline Conveyor (Figure 8) (M) handles both pallets entering the system for storage and pallets retrieved from the system going to the pick area. Pick Area Mainline Conveyor (R) delivers pallets to pick spurs. Conveyors M and R can be physically viewed as one continuous, moving conveyor.

Data zones

Mainline Conveyors M and R are artificially or logically broken up into 24 segments or data zones. Physically, there is no break or independent control of the zones, but they are logically used by the computer to map and control the movement of pallets through the conveyor network. The zones are delineated by sensing devices, either photocells or switches. A map of the zones is kept in the computer. If a pallet is in one of the zones, the corresponding entry in the computer map contains the associated data for the pallet—pallet number and destination (for input, aisle number; for output, pick spur number). Both the input and output buffers for each aisle have the three positions delineated by sensors for mapping in the computer.

Pallet tracking and mapping

As a pallet moves from one zone to the next, the sensing device is activated and a signal is sent to the computer. The computer determines from the signal which zones are involved and, also, the associated pallet and map entry. The computer updates the map by moving the pallet data from the zone (map entry) the pallet is leaving to the zone (map entry) the pallet is entering. By this means, the computer tracks the physical movement of the pallet through the network and knows the relative position of all the pallets on the conveyor. The relative positioning by zones occupied by the pallets is a key to the control scheme. Zone lengths average about 10 feet, with variations depending on physical characteristics of the conveyor. The computer knows to within a few feet the actual physical position of the pallet by knowing which zone it is in.

Movement from size sense

When processing of a pallet is complete at the size sense station, control is passed to the conveyor tracking programs along with the pallet number and destination. The computer sends a signal to release the pallet from the size station. If the pallet is marked for reject, a second signal is sent to divert the pallet onto the Reject Conveyor (D); the computer releases the pallet onto Spur S7 when there is a free space and when no conflict exists with pallets on conveyor R going to S7. The arrival sequence of the pallets is recorded in the map for Spur S7.

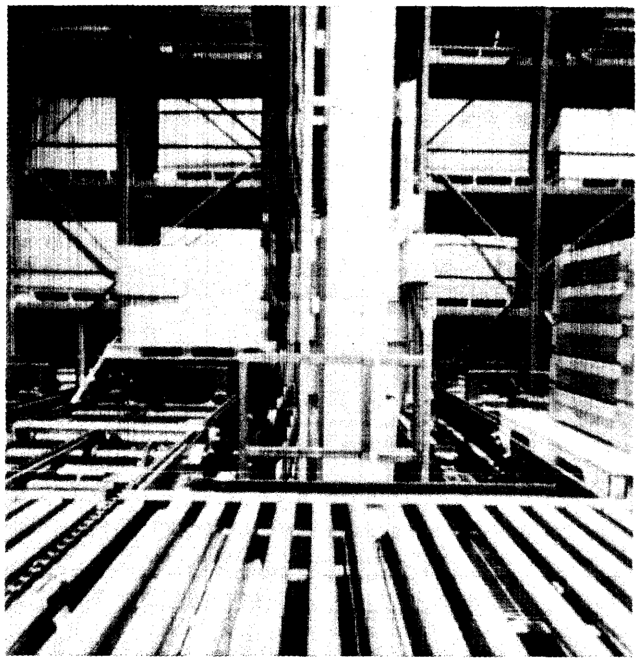


Figure 15—Cranes transporting a pallet

Accepted input pallets move into the system to flow on Mainline Conveyor (M). The pallets are tracked as they move through the zones.

Input-pallet diverting (Figure 14)

As an input pallet enters the zone containing the input-buffer transfer, the pallet data is checked to see if the destination matches the aisle being approached. If a match is found, the computer sends a signal to raise the pallet stop on the input-buffer transfer. When the pallet hits the stop, the chain transfer raises and moves the pallet into the first pallet position. Signals indicating start and completion of transfers are sent to the computer; these are used to determine failure conditions and to aid in control during merging of output pallets.

Input-buffer movement

As the pallets are indexed through the input-buffer spur positions, the pallet data is also tracked within the computer. When a pallet reaches the pickup position, a signal is sent to the system that the pallet is ready to be stored, and the pallet data is passed on to the programs controlling the cranes.

Output-buffer movement

The cranes retrieve pallets from storage and deposit them on the output buffer. When a pallet is placed on the deposit station of the buffer, a signal is received and the crane control programs pass data (pallet number, pick-spur destination, and sequence data) to the conveyor programs.

The pallet indexes and maps through the buffer positions toward the mainline conveyor as the positions become free. When the pallet is in the position closest to the mainline, it is ready for merging.

Output-pallet merging

The computer scans the map entries for the zones both upstream and downstream on the mainline from the merge point (output buffer) to determine whether there is enough free space on the mainline. Pallets are not merged if a pallet collision would occur. This is also true of subsequent transfers from segment M to segment R (Figure 8). The zones may be physically empty or "logically" empty.

A zone may physically contain a pallet that has started to divert into an input buffer, but it can be considered logically empty since the divert will be completed before the merge of the outbound pallet takes place. The use of logically empty zones effectively raises the capacity of the conveyor in pallets per hour.

When the needed free space is found, the computer sends a signal to start the merge. A signal is returned when the merge is complete; the data is then moved to the mainline map. The outgoing pallets on the mainline are tracked in the same manner as incoming pallets.

As pallets divert into aisles, free space is created, which is used by pallets merging out. This replacement basically allows the same rates of pallets in and out on a single conveyor as would be accomplished using separate conveyors for input and output, at the same speed of conveyor movement. Some loss of capacity occurs, however, over short periods of time with a single conveyor when the input and output distributions across aisles do not match, thus preventing total replacement.

Sequence control

Under certain circumstances, it is necessary for pallets to arrive at a pick spur in a certain sequence, that is, pallet 136 must arrive before pallet 35 in Spur S3. The pallets are requested one at a time sequentially for a pick spur. However, because of the physical nature of the system, they may not always arrive in the sequence requested without intervention. For instance, the first pallet requested may be on an aisle that has five output requests already pending (requests for a given aisle are serviced in the order received); the second request (perhaps generated two minutes later) is to an aisle that has no output commands pending. The second pallet will probably be serviced first and arrive at the pick spur first. A second example of this is when the first pallet requested is on Aisle #1 and the second is on Aisle #9.

When a pallet is ready to merge, a check on required sequencing is made. If sequencing is needed (another pallet must precede this pallet to the pick spur), the preceding pallet must have previously entered the mainline and be downstream of the merge point. If this condition is not met, merging is held until the condition is satisfied.

Pick-area tracking and diverting

Outgoing pallets are tracked along Mainline Conveyors M and R (Figure 8). As a pallet approaches the zone containing the transfer into a pick spur, the spur number is matched against the destination of the pallet. If a match is found, a signal is sent to divert the pallet into that pick spur. A signal is received when the pallet has completed the transfer into the spur, and the map of actual arrival sequence is updated. The actual sequence of arrival is used to control reporting in the pick area.

Conveyor rejects

If a pallet takes the wrong divert or misses a divert—as in the case of a stuck relay—the pallet is rerouted by the computer. If a crane goes down while a pallet is on the way to it, the pallet is also redirected. If these conditions occur on the crane buffers or on the mainline, they are routed to Pick Spur S7. Spur S7 handles the rejects from both the input and the mainline as well as pallets for picking. For any reject pallet going to S7, an appropriate message for the operator is printed on the 1053 next to the spur.

Failure trapping and recovery

Control of the conveyors is accomplished by monitoring events involving positive action, such as a pallet tripping a limit switch or breaking a photocell beam. Timing is not directly used in controlling the movement on the conveyors. However, timing is used to trap problems, and more importantly, potential problems on the conveyor. For instance, if it normally takes 12 seconds for a pallet to travel the length of a given zone and the signal of the pallet entering the next zone is not received in 15 seconds, a malfunction is assumed and the conveyor stopped. A flashing red light is turned on in the computer room, and a message is printed describing the type of failure and the location. These problems are checked and corrected by, for example, clearing a pallet jam or replacing a burned-out photocell. An appropriate physical recovery procedure is followed, such as moving the jammed pallet to the next photocell. The associated recovery code is then entered into a 2791, and the system restarts from the point where it stopped.

Immediate stoppage of the conveyor is intended to avoid major problems, such as would be caused by several pallets piling up behind a jammed pallet. This allows easier physical correction and makes logical recovery (ensuring that the physical condition matches the computer map) possible without the lengthy and costly procedures required to flush the pallets from the mainline, to ensure integrity of the system. Most failures can be corrected in five minutes; whereas, a flush of the mainline and restart of the system requires about one hour.

The detection of failure is done on the mainline, on all transfers, on the size sense station, and on input/output buffers.

Command selection

After successful completion of the previous crane command, the next command is selected if there is either a pallet for storage in the pick-up station or a request pending to retrieve a pallet. If only one of these commands exists, it is done next. If both exist, a selection must be made to either store or retrieve next. The requests for retrieval are placed in a list for each crane as they occur and are serviced in the same order. Communication with cranes can take place anywhere in the aisle once the previous command has been completed. The crane remains at the position of completion of the last command.

Thus, the last command has a considerable influence on the selection of the next command because of the position of the crane. If the last command was a retrieve, the crane is at the head of the aisle as a result of dropping off the pallet; a store will probably be done next since it does not involve crane movement to pick up the pallet. If the last command was a store, the next will probably be a retrieve, since the crane is already down the aisle. In this manner, the normal operation will consist of alternate stores and retrieves, the stores/retrieves being effectively paired. This limits unnecessary, unproductive movements of the crane, that is, movement when the crane is not carrying a pallet (time is really the factor limited).

This pairing is logical. Commands are sent individually; "dual" commands (where both the store and retrieve commands are physically sent to the crane at the same time) are not used. Dual commands make practical error recovery impossible, lessen flexibility, and do not gain any advantage in capacity when the crane can be communicated with anywhere in the aisle.

Under certain circumstances, this pairing may be overridden. For instance, the last command may have been a store just several bays down the aisle. A retrieve just requested may be at the end of the aisle. If a pallet is waiting for storage, another store command is given; this will further limit unnecessary movement. If a large number of retrieves are pending and there are only a few stores, the retrieves will be given priority to help balance the system.

Slot selection

If the command selected is a store, the actual location must be selected. This selection is dependent on the height of the load. The aisle was chosen at the input. If no retrieves are pending, the slot closest to the front of the system is selected to minimize the time required for the command.

Since vertical and horizontal motions are performed at different rates, each portion of the command must be considered and balanced to select the best slot. For instance, a slot one vertical level above and five horizontal bays from the pickup station is closer in terms of time than one five vertical levels above and one horizontal bay from the pickup station. The horizontal and vertical motions and the associated times are the key elements in minimizing total time from the present or future anticipated points.

If there is a retrieve command pending, the store slot is chosen relative to the retrieve slot. Ideally, the store slot would be selected exactly opposite the slot for the next retrieve, so that unnecessary motion and time would be totally eliminated This elimination has more significance than may be anticipated. The movements in the last ten feet and in the final positioning are slow compared to the rest of the motion and comprise a large percentage of the total command time. For example, if the distance is doubled from 100 to 200 feet, the travel time is increased by only 20 percent.

Elimination of the final positioning on one of the commands substantially reduces the total time for both commands. Selection of a storage slot exactly opposite the slot for the next retrieve is, of course, impossible most of the time, but the correct choice can minimize the unproductive time. The empty slot that minimizes movement time relative to the retrieve location is therefore chosen.

This elimination and minimization substantially raises the effective capacity of the cranes over the nominal capacity obtained by random selection of the storage location. It also increases the response and improves the balancing of the system and aids in overcoming downtime of the cranes.

Command transmissions and execution

Once the command has been selected, it is formatted and sent to the crane. The command describes crane number, type of command (store, retrieve), and location (horizontal, vertical, left/right). Once the crane accepts a command, it is no longer under direct control of the computer. The local logic on the crane directs the crane until it completes the command or detects an error.

Command completion

When the command is successfully completed a signal is sent to the computer, and the location file is updated to show whether the slot is full or empty. The pallet file is also updated to show the pallet is in transit to the pick area or is in storage, with the new location.

Error detection and recovery

If an error is detected, an error-alert signal is sent to the computer. The computer, in turn, formats a special sense command and sends it to the crane. This causes the crane to return status data to the computer. The status data describes the error(s) detected and indicates whether the pallet is on board the crane. From this data, it can be determined whether the error is recoverable. If it is recoverable, a new command to finish the operation is formatted and sent to the crane. This new command is based on the previous command, load-on-board data, and where the failure occurred in the command.

This may be the same or a different command. When a retrieve is being performed and the error indicates that no pallet is on board, the pallet has not yet been removed from the slot and the retrieve command is reissued. If the pallet is on board, it has been removed from the slot. The command is then changed to a store, designating the deposit station on the output buffer as its store address.

If the error is not recoverable, as in the case of an overhanging load, the flashing red light is turned on, and a message describing the problem is printed.

The computer takes the crane logically offline to prevent further use until corrective action has been performed. The pending requests for output are cancelled so that output is not totally stopped while waiting for a sequenced pallet from the crane that is not operating.

When the problem is corrected, the crane is reactivated to the system via a 2791 transaction.

At the end of the daily one-shift picking operation, the picking area is cleared and all pallets are returned to storage. Approximately 20 minutes before the end of the shift all of the active batches are terminated by the pick-area technician so that no new work is generated by the system. Just before the end of the shift, the workers release the pallets for re-storage. Any unfilled requisitions are included in the batches for the next day.

SYSTEM OPERATION—BATCH REPORT

A number of reports in various categories are produced to aid in monitoring and managing the system. Some of the major reports are discussed below.

Physical activity

Crane & Conveyor Activity—Several reports are produced showing stores and retrieves done by each crane, the number of good commands and number of errors by type, the number of offline conditions by crane, and the total time each crane is inoperable during the period covered by the report. The conveyor reports cover all malfunctions on the conveyor by device for both errors which stopped the conveyor and errors filtered by the software. The total up time is reported.

Pick-Area and Input-Area Activity—These daily reports show the number of pallets passing the input station by tape such as new, recycle, parent, and sample. The number of pallets processed in each pick spur are shown by type such as count, requisition, no-action, and unexpected. These reports are not used to measure people, but rather to show activity trends that may reveal potential problems.

Storage Utilization—Periodically a report showing a percentage of slots used by size, by aisle, and by total system is produced. This is used to predict either over- or under-utilization of the system storage.

Inventory status

Part-Number Inventory
Pallet Inventory
Pallet Inventory in Slot Sequence
Aisle and Part-Number Inventory

This provides a listing for specified part numbers for all the pallets in slot sequence. This is used to audit and count part numbers that have many pallets in the system, rather than retrieve the parts for counting. This is done particularly with part numbers where each pallet contains a quantity of one.

Auditing and tracing

Daily Pallets In and Out—This records all pallets that are new to the system for the day and those that have physically left the system (zeroed out). This provides additional tracing of movement of any particular pallet.

Pallet in Transit—As pallets move through the system (from input to storage, in storage, from storage to pick, from pick back to input for recycle), the transit status is changed. Since the conveyor network is cleared at the end of the working shift, no pallets should appear on the in-transit report. They should be either physically out of the system or be in storage. Any pallets on the report are flagged as errors, and a physical check of the warehouse areas is made to find them.

ASRS Audit—The receiving documents, requisitions, and count cards are compared to the transactions that have been keyed in through the terminals; any discrepancies are noted and investigated. This makes it possible to correct common errors such as keying in the quantity 90 instead of 9.

System Error—This report shows all potential errors or errors trapped during realtime operation such as unexpected full bins, unexpected empty bins, or quantity discrepancies discovered in the picking area. Each item is audited by the warehouse personnel.

Inventory activity/management

Bin-Lock—This is a report of all pallets that are bin-locked, under either inspection or management lock, showing the number of days they have been locked.

Part-Number Activity—This report lists all part numbers and shows the number of times each part number has been accessed during that reporting period.

Inactive Part Number and Pallet—This shows any part number and/or pallet that has not been accessed during a management-specified period of time. This, in conjunction with a part-number activity report, is used to determine whether the correct parts are stored in the stacker-crane system.

File recovery

A set of programs rebuilds the files to the current status using the transaction-tape records and previous disk files so that the data is not lost if files are accidentally destroyed during realtime operation.

CONTROL SYSTEM—MAJOR FEATURES

The following is a recapitulation of the major features in the system:

- Control of nine individual cranes
- Closed-loop conveyor system
- Total pallet tracking throughout the conveyor network
- Handling of both inbound and outbound pallets on the same conveyor
- Partial sequencing of outbound pallets
- Ability to handle up to 133 pallets in and 133 pallets out per hour
- Detection of crane/conveyor malfunctions, with dynamic restart capability
- Automatic size sensing
- Storage of variable-size loads in five different-size slots
- Both dynamic selection of slots for storage and selection of pallets for retrieval
- Storage and accounting of pallets in 15,600 individual slots
- Accounting for 8,000 part numbers
- Handling of multiple pallets for a part number
- Handling of two parts per pallet
- Ability to perform partial- and full-load picking
- Recycling of partially picked pallets
- Ability to perform counting functions
- Operator guidance for all man/machine interfaces
- Dynamic recording of all activity on the pallets and parts, with realtime update of data with appropriate auditing
- Ability to logically lock pallets for inspection, counting, or management review
- Emergency requests for parts for picking or resample
- Dynamic assignment and reassignment of pick-area work
- Tape backup for disk-file updates and file-rebuild capabilities

Benefits of computer control

Total computer control of the ASRS with the 1800, is the most technically feasible and most economical control system available, considering the volume and complexity of operation. The online 1800 control system was compared with two other possible methods:

- Manual Control—all control done by people in the system; this includes manual recording and updating of data.
- Semiautomated—extensive vendor control logic to aid people in controlling the hardware, with data maintained offline on a small disk computer.

Online control is economically superior to either of the other alternatives in both initial investment and continuing operating cost. The major advantage in initial investment is realized in the reduction of vendor-provided control hardware.

Some of the major benefits derived from total computer control of the ASRS are briefly discussed in this section.

- High-Volume Handling—The ability to handle a high flow of pallets in the conveyor network and on the cranes. Better handling of peaks of activity, which are natural in an ASRS environment.
- Reduced Hardware Investment—Reduction of vendor-provided hardware, especially for control, such as:
  - Card readers on the cranes
  - Shift-registers on the conveyors
  - Separate conveyors for input and output to the ASRS (this also eliminated construction costs)
- Improved Efficiency—Significant improvements in efficiency, particularly in control, data recording, and auditing.
- Improved Space Utilization—Accomplished by:
  - Dynamic update of slots being freed for immediate reuse
  - Easy handling of multiple parts per pallet
  - Resizing and reselection on recycles
  - Pulling smallest quantity first
- System Integrity—Reduction of exposure to "lost" loads and data errors as a result of:
  - Elimination of many repetitive manual decisions
  - Elimination or reduction of data recording and transcription
  - Forced total reporting
  - Online auditing of data
- Better Exception Handling and Increased Flexibility—Increased ability to dynamically meet changing requirements of the warehouse, in order to meet their customer needs without wasted effort, by easily handling:
  - Emergency requests
  - Bin-locking
  - Hardware failures
  - Changes in work assignment
- Effective System Usage—Smoother and more effective use of the system and the hardware as a result of:

Control decisions made from a total-system viewpoint
Matching of crane commands and slot selection
Immediate trapping of potential hardware problems, with dynamic recovery

APPENDIX I

Disk files

In order to utilize and support the use of automated cranes and conveyors, it is necessary to maintain a current picture of all inventory in the system. This is done by updating the status of a pallet, part number, slot location, or request for output, as a change occurs. To accomplish this, it is necessary to maintain six basic files: the pallet file, part number, location, parent, requisition work queue and work queue directory.

The part number file contains records defining all inventory in the system. Each part number authorized to be stored in the ASRS has one unique record which summarizes the total quantity and status of that part. Also included are pointers to pallet records, which contain additional information about the associated part number. Part-number history and statistics are also accumulated on the record, such as the date of last activity against the part number and a counter of the number of requests for pallets containing the part number.

Each physical ASRS pallet has a unique corresponding pallet record. Pallet records are not created or deleted when pallets enter or leave the ASRS. instead the status is updated to reflect its current location and availability. Pallet records are used to describe the specific characteristics and status of the parts on the pallet. This includes actual location, quantity, part number(s), current status (in-storage, empty, in transit, etc. . . .), and pointers to other pallets that contain the same part number. In this way, pallet records of like part numbers are chained together, with pointers of the first and last pallets in the chain contained in the part-number record.

The location file provides an indication of which slots are full/empty. It does not include which pallets or part numbers are stored in a particular slot. The file is organized by aisle, level, and size of slot to provide easy access to available information.

The parent file, like the part-number file, points to chains of pallet records. All pallets in the chain contain parts that are currently being inspected. That is, a sample is taken from newly received parts and inspected, while the balance of the parts (the parent) are stored in the ASRS unavailable for requisition filling until the sample has passed inspection. When the accepted sample enters the ASRS, the associated chain of parent pallet records is added to the part-number chain. The part-number record is then updated to reflect the availability of the accepted parts. Purchase-order number and shipment number are also used as qualifiers, to link samples and parents of like part numbers. In this way, a sample will only release those parts that were contained in the original shipment.

The requisition work queue (REQWQ) provides a summary record (macro) of the total requirements against a part number for a given day. All normal requisitions to be filled are sorted by part number and the gross quantity requirements are maintained on a REQWQ record. The record is then assigned a unique sequential number—the macro number. These macros will then be exploded into specific requests for pallets when needed. In this manner, changes in the real time environment will be taken into account.

A second file, the work-queue directory (WKQDR), indexes the REQWQ macros and groups them into "batches of work." Each work-queue directory record has a starting and ending macro number, and the number of the next macro to be "worked on." The pick-area technician can now assign or activate any given "batch of work" and the macros

Basic Operating Files

	Size			
Name	REC/File	Words/Rec	Organization	Information
Pallet (PALET)	20,020	24	DIRECT-Sequential by Pallet Number	Pallet number, status, location, part number, P/N status quantity, date of entry, date of last transaction, next pallet pointer, previous pallet pointer, entries for 2nd part number.
Part Number (PART #)	10,440	11	Index Sequential By Part Number	Part number, status, quantity first pallet pointer, last pallet pointer, date of last transaction activity counter.
Location File (LOCFL)	9	320	Sequential by aisle number	Aisle number, number of full locs for size, number of locs Size N, SLOT status (empty or full), error log by crane number.
Parent File (PARFL)	1,050	9	Indexed Sequential by PO & Ship & P/N	Part number, purchase order number, shipment number, first pallet pointer, last pallet pointer.
Requisition Work Queue (REQWQ)	1,050	9	Sequential by Part Number	Macro number, part number, quantity, first pallet, RIC count, status.
Work Queue Directory (WKQDR)	80	4	Sequential by Batch Number	Starting macro number, ending macro number, status, next macro pointer

associated will be processed by the computer. This, in turn, will generate specific requests for pallets until the macro is completely satisfied or all existing parts in the ASRS are exhausted.

IBM 1800 software

Multiprogram Executive Operating System (MPX Version 3) is used unmodified for the 1800 computer operating system. This system provides all of the non-application software support. It has the required capabilities to service:

- Interrupts
- Multiprogramming Levels and Priorities
- 2790/1053 Terminals
- Disk and Tape Files

The application programs are written in both 1800 Assembler, Fortran, and 2790 Macro Languages. These are mixed within modules and individual programs. The crane and conveyor control programs are predominately Assembler. The other areas are predominately Fortran and 2790 Macro. The number of installed programs by area are shown below:

Real Time	
Conveyor Control	22
Crane Control	12
Pick Area	31
Input Area	32
File Support	18
Miscellaneous	4
Batch	42
Total	161

Additionally, over 60 uninstalled programs were written for test purposes.

Supporting government cost planning of industrial wastewater treatment

by EDWARD H. PECHAN, RALPH A. LUKEN and JACOB E. MENDELSSOHN

U.S. Environmental Protection Agency
Washington, D.C.

INTRODUCTION

The 1972 Federal Water Pollution Control Act Amendments established a series of national goals with the purpose of reducing and eliminating water borne pollution in the United States. One of these goals requires the implementation of best practicable control technology by all industrial dischargers by 1977.

The purpose of this paper is to describe a computer based system designed to estimate the costs of achieving these Federal Standards on non-thermal discharges by industry. The costs are estimated for each of 250,000 establishments in 14 major industry groups affected by the new standards. The computer system can then aggregate these results to the desired level of detail, supporting management decisions at federal, regional, and state levels for any of the 14 industry groups or their subsets.

A major purpose of the development of the Industrial Cost Model is to produce national cost estimates by industry for inclusion in the *1973 Economics of Clean Water* report to Congress. Thus, one goal of the models developed was to attempt to achieve as high a level of compatibility as possible with previous Economics of Clean Water reports. The other goals of the model and the constraints under which it was developed are discussed in more detail in a later section of this paper.

The model was developed by EPA Headquarters in Washington, D. C. However, the information from the model is available to state and local government institutions as well as to selected private sector organizations. It is hoped that the full utility of the system can be taken advantage of.

Later sections of the paper discuss the concepts underlying the model development, an overview of the model itself and some summary data and suggested applications for the model results.

CONCEPT OF THE MODEL

Some of the basic outward features of the Industrial Model were mentioned briefly in the introduction. It is the purpose of this section to discuss the development of the model from the standpoint of the goals to be achieved by the model and the information and resources available to support the model's development.

Model goals

The major goals of the model as seen during its planning stages were as follows:

- To produce information on the cost impact of 1977 Federal standards on industry, by industry type, for publication in *1973 Economics of Clean Water*.
- The most current available data on water use, industrial plants, and costs of control alternatives were to be used.
- To support economic impact analyses to be made by industry type.
- To produce cost summary reports in breakdowns other than major industry type* including EPA region** and state, as aggregates or by major or minor industry type.
- To achieve compatibility where possible with the previous versions of *Economics of Clean Water*, particularly the 1972 edition.
- All of the existing industrial sources in those industries under study were to be represented in the total cost figures developed.***

While the goals above refer to specific information outputs, it is important to realize that the study focused on producing results which would be meaningful as management tools. Some examples of how the model results can be used at various levels of government and by the private sector are discussed in a later section.

* Reference to major industry type or industry type is to 2 digit Standard Industrial Classification (SIC) code. Minor industry type refers to 3 or 4 digit SIC code.

** The 10 EPA regions are shown in Figure 1.

*** This ruled out use of the EPA Refuse Act Permit Program (RAPP) system and successor which has detailed information on industrial sources but contains only a small fraction of the total number of sources.

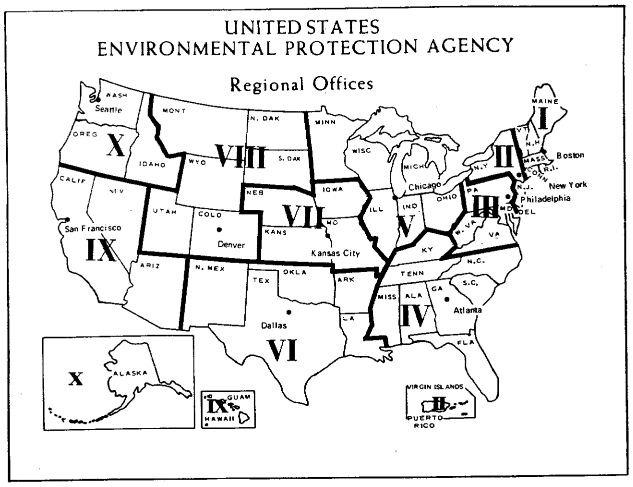


Figure 1—EPA regions

Model constraints

The resources available for the project were limited considering the scope of the goals to be achieved by the project. The effort was restricted to using information and data currently available to the Agency; there was not adequate time to undertake any sort of large data collection effort.

Some software was available from the Industrial Model developed for the 1972 Report but it had been developed for a different computer system. In addition, the formats and contents of the input files had to be changed significantly. Thus, although the existing programs were available, those which were used had to be almost completely rewritten.

The time constraint primarily served to limit the number of detailed reports which were produced by the system and the thorough analysis of those which were. The computer runs for the project were being made during a time of both hardware and software difficulties by the Agency computer services contractor. Most of the major runs had a large time requirement and were submitted for overnight turnaround. However, the various computer difficulties often dictated more than one attempt at each run.

Overall, the project team was able to meet the final deadlines for production and analysis of the model and its outputs. In addition to the summary information published in *Economics of Clean Water*, there is a variety of other more detailed information which could be useful to management and planners at state and local levels.

MODEL OVERVIEW

Several steps are required to achieve the desired calculation of costs of meeting Federal standards in 1977. The steps required are shown below:

1. Compute the cost of abatement facilities to meet 1977 standards on the current stock of plants (i.e., retrofit costs).
2. Apply growth factors to compute the costs of the 1977 standards on the plants to be built between now and 1977.
3. Compute the replacement cost of all abatement facilities adequate to meet 1977 standards on existing plants.

The costs represented by calculation 1 are the total investment required for existing plants. The costs represented by the difference of 1 and 3 represent the new retrofit investment required by 1977. Finally, the costs represented by calculations 1 plus 2 minus 3 is the total to be invested by 1977 including new plants.

In addition to the capital (or fixed) costs required, the annual operations and maintenance (O&M) costs for the facilities is also computed. Knowing capital and O&M costs, a "total annual cost" (TAC) can be computed assuming an annualizing factor to convert capital cost to an annual charge.

The industries covered by the study (by 2 digit SIC code) are shown below:

SIC	Industry
20	Food and Kindred Products
22	Textile Mill Products
24	Lumber and Wood Products
26	Paper and Allied Products
28	Chemicals and Allied Products
29	Petroleum and Coal Products
30	Rubber and Plastic Products
31	Leather and Leather Products
32	Stone, Clay, and Glass
33	Primary Metals
34	Fabricated Metals
35	Non-Electric Machinery
36	Electric Machinery
37	Transportation

Figure 2 shows the overall flow of the industrial cost model system. This series of program and files is executed for com-

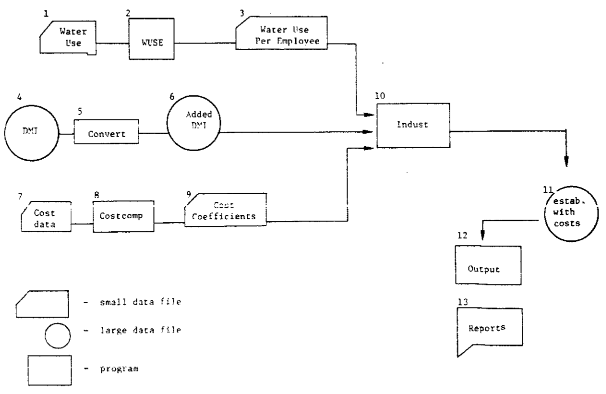


Figure 2—Industrial cost model flowchart

putations 1 and 3 given above. Computation 2 and the various difference calculations are performed by hand.

The blocks of Figure 2 are explained below. All programs were written in FORTRAN IV.

1. *Water Use Data.* This is basic data taken from the report *Water Use in Manfuacturing** which indicates the water used by minor industry types in 20 water use regions.** Table II from *Water Use in Manufacturing* provided the following detail required for the model.***

—Water intake by purpose, Gross Water Used, and Water Discharged: 1968
  A—Industry Group and Industries
    employees
    water intake
    process water intake
    gross water used
    total water discharged
—Water intake by purpose, Gross Water Used, and Water Discharged: 1968
  B—Water Use Regions, Major Industry Groups, and Industries
    employees
    water intake
    process water intake
    gross water used
    total water discharged

2. *WUSE Program.* This program converts the basic data from *Water Use in Manufacturing* to data by minor industry type and water use region. The model operates as follows:†

The model forms ratios of regional to national totals to define variability between major classes of manufacturers geographically and also looks at the variability between industrial classes within a major manufacturing class.The water use per employee values define the process water per employee coming into contact with the product stream as follows:

$$WT4R_{(e)} + \frac{WP4}{E4} \times WP2R' \times WU2R' \times \frac{WU4}{WI4}$$

Where the process water to be treated is constrained to be no greater than the total amount of water discharged.

The parameters are defined as follows:

E—employment
WI—water intake
WP—water intake specifically used in product processing
WD—water discharged
WU—water used (gross used in plant including recirculation and reuse)
WT—total water in contact with product stream

Also,

4—four-digit SIC Code
2—two-digit SIC Code
R—water use region designator—(If no R exists the parameter is assumed to be a national value)
(e)—designates a per employee value
primes—represent ratios calculated between regional and national values

The following calculations are performed to yield the result above:

(a) process water per employee ratio; regional defined ratio at the major industrial class level

$$WP2R' = \frac{WP2R/E2R}{WP2/E2}$$

(b) recirculation ratio; regionally defined ratio at the major industrial class level

$$WU2R' = \frac{WU2R/WI2R}{WU2/WI2}$$

(c) water discharged per employee: represents the maximum water to be treated per employee for each industrial category, regionally defined

$$WD4R_{(e)} = \frac{WD4}{E4} \times \frac{WD2R/E2R}{WD2/E2}$$

(d) process water per employee coming into contact with the product stream defined by industrial category and region

$$WT4R_{(e)} = \frac{WP4}{E4} \times WP2R' \times WU2R' \times \frac{WU4}{WI4}$$

Where:

$$WT4R_{(e)} = WD4R_{(e)}$$

If:

$$WD4R_{(e)} < WT4R_{(e)}$$

Since the computations above were based on data which represented water use policies of some time ago (1968), the program produced six different water use scenarios based on different improvements in water use efficiency. The details of these scenarios is not repeated here.* One scenario was selected as most appropriate and used in most computations. Outputs from other scenarios are available to test model sensitivity and validity.

3. *Water use per employee.* This file contains the water to

* *Water Use in Manufacturing,* A report from the *1967 Census of Manufacturers,* U. S. Department of Commerce Report No. MC67[1]-7, Data are for 1968.

** Only seventeen regions are used in the model. Cumberland is confined with Tennessee, Alaska with Pacific Northwest, and Hawaii with California.

*** From *The Economics of Clean Water,* Volume II, Environmental Protection Agency, 1972.

† From *The Economics of Clean Water,* op cit.

* See *The Economics of Clean Water,* Environmental Protection Agency, 1973.

be treated per manufacturing employee by major and minor industry type and by 17 water use regions.

4. *Duns Market Indicators* Extract.* The following data elements were extracted from the Dun's Market Indicators (DMI) file for each of the approximately 250,000 establishments with the desired SIC codes.

- SIC code
- manufacturing employment
- state code
- county code
- SMSA code

5. *Convert Program.* The convert program was used to add two fields to the DMI extract. The fields added were:

- water use region (based on state and county codes)
- EPA region (based on state code)

6. *Added DMI.* The DMI extract file that contains the new fields discussed above.

7. *Cost Data.* The basic cost data were determined for the following twelve types of treatment

Oil Separation	Aeration
Equalization	Natural Stabilization
Coagulation	Chlorination
Neutralization	Evaporation
Air Flotation	Incineration
Sedimentation	Activated Sludge

The basic cost data were taken primarily from a report by an EPA contractor.** Three or four points were available for each treatment type based on the costs for different water flows.

8. *COSTCOMP.* The cost points were converted to coefficients of equations of the form

$$\mathrm{LOG(COST)} = \mathrm{A} + \mathrm{B}[\mathrm{LOG(FLOW)}] + \mathrm{C}[\mathrm{LOG(FLOW)}]^2$$

where

LOG	= base 10 logarithim
COST	= cost in millions of dollars
FLOW	= flow in millions of gallons per day
A,B,C	= coefficients from COSTCOMP program

Both linear (coefficient C=0) and quadratic (coefficient C non 0) regressions were run on the basic cost data. The type of equation with the highest F test value was selected. In some cases, the curve was divided into separate equations based on level of flow.

9. *Cost Coefficients.* The values of A,B, and C for each of the twelve treatment types (and segments) for both capital (fixed) and operating and maintenance costs.

10. *INDUST, Cost Calculator Model.* This is the key program in the system. It operates on each of the establishments from block 6 and, using files from blocks 3 and 9 and other data described below, produces the treated water flow and capital and O&M cost requirements in 1972 dollars for each of the twelve treatment types which may be imposed. In addition, a filter is applied which eliminates all establishments with less than one million gallons per year of treatable wastewater since these establishments are undoubtedly either using municipal facilities or applying wastes to land; 100,000 establishments were rejected in this way.

The inputs are:

- water use per employee (block 3)
- list of establishments to model (block 6)
- cost coefficient (A,B, and C) of the form:

$$\mathrm{LOG(COST)} = \mathrm{A} + \mathrm{B}[\mathrm{LOG(FLOW)}] + \mathrm{C}[\mathrm{LOG(FLOW)}]^2$$

(block 9)

- factors for accommodating differences in O&M costs for each state
- factors for estimating the capital cost differences for each water use region.
- the percentage of plant water use that is treated by each of the various treatment processes. This information was compiled for each SIC code considered.
- average operating days per year for each SIC.
- waste strength scale factor relative to municipal wastes of the form SF=0.4+0.6× (waste strength relative to municipal waste)
- where SF is greater than or equal to 1.0
- a factor for accommodating differences in the cost of the same effluent treatment process in different SIC industries.

11. *Establishments with Cost.* This file consists of a subset of the added DMI file (block 6). The subset is all establishments with a computed water usage greater than one million gallons per year (equivalent to the domestic waste from thirty people). The file contains all of the elements from block 6 and the following additions

- the annual process water requiring some type of treatment
- the capital cost of treatment facilities for each of the twelve treatment types
- the operating and maintenance costs for each of the twelve treatment types.

12. *Output Program.* This program performs data aggregations and computes summary statistics for the aggregated groups. Before running the program, the output file from block 11 is sorted on the appropriate fields. The following

* Computer file maintained by Dun and Bradstreet Inc. and available to EPA under contract.

** Associated Water and Air Resources Engineers, Inc., "Estimating Water Pollution Control Costs from Selected Manufacturing Industries in the U.S., 1973-1977", Part I, June, 1973.

```
                                         EPA REGION     2
                           REGIONAL REPORT BASED ON AT LEAST MEDIAN 1968 FLOWS
                                 1968 FLOW CONDITIONS ( 1972 DOLLARS)
   TOTAL NUMBER OF MANUFACTURING PLANTS... 18079

                              MAXIMUM           MINIMUM           AVERAGE            TOTAL              S.D.
   MANUFACTURING EMPLOYMENT      8000.                1.               83.          1506917.              300.
   FLOW (MGY)                   11460.                1.               33.           599670.              239.
   TOTAL CAPITAL COST         8604835.             3278.            63655.        1150825980.           200920.
   TOTAL ANNUAL O&M COST       758996.              254.            10079.         182214976.            12349.
   TOTAL ANNUAL COST          1341676.             3092.            18159.         328293888.            33147.

                                       ANNUAL COST OF WASTE TREATMENT
                              C&M          REPLACEMENT           INTEREST            TOTAL

                              182214976.      57541280.          88613552.         328369664.

                                   DISTRIBUTION OF OPERATION AND MAINTENANCE COST
   COST RANGE                MAXIMUM           MINIMUM           AVERAGE            TOTAL              S.D.    NUMBER
   >  500000.                 758996.           758996.           758996.           758996.               0.       1.
   100000. TO  500000.        436688.           104813.           210022.          4200437.          113971.      20.
    50000. TO  100000.         96494.            50244.            68462.          2875384.           13087.      42.
    10000. TO   50000.         49790.            10099.            16037.        152403056.            4202.    9503.
        0. TO   10000.          9995.              254.             2585.         22003632.            1406.    8513.

                                     DISTRIBUTION OF INVESTMENT REQUIREMENTS
   COST RANGE                MAXIMUM           MINIMUM           AVERAGE            TOTAL              S.D.    NUMBER
   > 5000000.               11616530.          5356611.          8372392.         66979136.         2436192.       8.
  1000000. TO 5000000.       4922698.          1020886.          1958778.        233094592.          899251.     119.
   500000. TO 1000000.        987717.           502355.           702241.        117976560.          141960.     168.
   100000. TO  500000.        499110.           100118.           182318.        460900864.           81895.    2528.
        0. TO  100000.         99969.             3934.            37792.        576547584.           20046.   15256.

                            DISTRIBUTION OF TOTAL CAPITAL COST FOR TREATMENT ($ 1000)

           *1*       *2*      *3*       *4*       *5*       *6*      *7*      *8*       *9*     *10*     *11*     *12*

 MAX     2006.     1308.    2790.     4373.      304.      684.    1276.    2241.       25.    2965.     550.    1081.
 MIN        0.        1.       0.        0.        3.        0.       0.       1.        0.       0.       0.       0.
 AVE       31.        8.      11.       37.       13.        7.      62.      23.        2.    1628.     124.     160.
 TOT    30437.    93235.  137512.   556432.    29867.    57515.    3904.  219937.    11062.    4885.    1112.    5267.
 SD        61.       26.      54.       99.       16.       18.     166.      73.        2.    1382.     169.     256.

                           DISTRIBUTION OF ANNUAL O&M TREATMENT COSTS ($ 1000)

           *1*       *2*      *3*       *4*       *5*       *6*      *7*      *8*       *9*     *10*     *11*     *12*

 MAX       56.       56.       5.        0.        3.       42.      34.     151.        0.     435.     641.      73.
 MIN        0.        0.       0.        0.        0.        0.       0.      10.        0.       0.       0.       0.
 AVE        1.        1.       1.        0.        1.        5.       6.      11.        0.     193.     280.      18.
 TOT     1068.    10748.   16351.     5239.     2274.    36280.     404.  105967.      176.     578.    2522.     602.
 SD         3.        1.       1.        0.        1.        3.       5.       4.        0.     210.     167.      15.

 SD  OR S.D.  REPRESENTS STANDARD DEVIATION.   THE DATA DISTRIBUTION USED IS HIGHLY SKEWED.
```

Figure 3—Sample output report

types of aggregations may be generated

- state
- EPA region
- industry type (2,3, or 4 digit SIC)

The third category, industry type, can be generated by itself (for national totals) or in conjunction with state or regional breakdowns (for industry within state or region).

Based on the aggregation level chosen, the number of entries, maximum value, minimum value, average, total, and standard deviation are produced for the following values:

- manufacturing employment
- basic total treatable flow
- total capital cost for structural cost (not including land)
- total annual operations and maintenance cost (O&M)
- total annual cost including O&M, interest, and replacement of capital
- annual O&M costs for five groupings of cost
- capital costs for five groupings of cost
- annual O&M costs for each of the twelve treatment type
- capital costs for each of the twelve treatment types

Also, total annual costs are broken down into O&M, capital replacement, and interest charges.

12. *Output Reports.* Figure 3 shows one page from an output report as described in block 11.

To compute the value of facilities in place, the Added DMI file (block 6) is replaced by a summary of plants with facilities derived from *Water Use Manufacturing.* The output is supplemented with data from an annual survey of pollution control expenditures.*

MODEL APPLICATIONS

This section discusses the variety of potential applications at federal, state, and local levels for the information produced

* McGraw-Hill Publication Company's Survey of Pollution Control Expenditures.

TABLE I—Percentage of National Totals by States

State	Capital cost of industrial water treatment*	Total industrial capital expenditures†	Annual cost of industrial water treatment*	Value added by Manufacturer†
	(Percent)	(Percent)	(Percent)	(Percent)
Alabama	1.8	1.7	1.8	1.4
Alaska	.2	.1	.4	.0
Arizona	.4	.5	.2	.4
Arkansas	.8	.8	.9	.7
California	7.6	7.7	5.7	8.9
Colorado	.7	.7	.8	.6
Connecticut	1.1	1.7	1.2	2.4
Delaware	.4	.3	.3	.4
District of Columbia	.0	.0	.1	.1
Florida	2.2	1.4	2.2	1.4
Georgia	2.3	2.3	2.4	1.8
Hawaii	.3	.1	.3	.1
Idaho	.5	.2	.6	.2
Illinois	6.4	6.7	6.2	7.4
Indiana	2.8	5.2	2.6	3.9
Iowa	.9	1.2	1.1	1.2
Kansas	.6	.5	.7	.8
Kentucky	1.1	1.5	1.1	1.4
Louisiana	3.2	2.5	2.8	1.1
Maine	1.0	.5	1.0	.4
Maryland	1.0	1.4	1.1	1.4
Massachusetts	2.2	2.2	2.6	3.1
Michigan	4.9	6.2	4.9	6.6
Minnesota	1.3	1.4	1.6	1.6
Mississippi	1.0	.8	1.0	.6
Missouri	1.6	1.3	2.0	2.2
Montana	.5	.1	.6	.1
Nebraska	.4	.4	.5	.5
Nevada	.1	.1	.1	.1
New Hampshire	.5	.4	.5	.3
New Jersey	4.2	4.2	4.3	4.7
New Mexico	.3	.1	.3	.1
New York	6.1	7.2	7.4	9.3
North Carolina	2.5	3.4	2.6	2.7
North Dakota	.1	.1	.2	.1
Ohio	6.9	8.9	6.3	7.9
Oklahoma	.6	.6	.8	.5
Pennsylvania	7.8	6.9	7.2	7.3
Rhode Island	.5	.4	.6	.5
South Carolina	1.4	1.6	1.3	1.2
South Dakota	.1	.0	.1	.1
Tennessee	1.7	2.4	1.7	2.0
Texas	9.0	6.3	6.9	4.2
Utah	.5	.2	.5	.3
Vermont	.2	.2	.2	.2
Virginia	1.4	1.7	1.5	1.6
Washington	2.9	1.4	3.4	1.7
West Virginia	.7	.9	.8	.8
Wisconsin	2.6	2.1	3.0	2.7
Wyoming	.2	.0	.3	.1

* From *1973 Economics of Clean Water.*
† *Statistical Abstract of the United States.* Department of Commerce. 1972 (1969 values).

by the Industrial Cost Model. The section is divided into two parts, the first discusses some of the applications to which the model can be put, the second presents an example of how the model outputs were used to perform a relative impact analysis for states and EPA regions.

Potential applications

One of the benefits from the methodology used in developing the Industrial Cost Model is the variety of aggregations which can be performed. The flexibility of uses for the model outputs are considerably enhanced by this capability of the model. At the present time, aggregations have been produced by EPA region, state, and major industry type. The model can easily produce additional aggregation by minor industry type, SMSA, or counties. Of course, the less aggregated the analysis is, the more likely that it may vary from the actual situations.

The mechanisms used in producing the control cost estimates are designed to account for the specific differences in control costs for each plant based on such factors as manufacturing plant type, water use, state difference, for operations and maintenance costs, and regional differences for capital costs. However, each plant has a certain amount of latitude in controlling water pollution. For example, there is considerable flexibility in the specific set of treatment types which are possible and there are additional tradeoffs between capital costs and operations and maintenance costs. The model outputs are based on estimations of the most likely overall abatement measures to be adopted; however, these measures will not necessarily apply to each specific plant. Thus, discrepancies between the model estimates and actual occurrences are more likely to be noticed at low level aggregations. Conclusions based on specific model outputs may be affected by this phenomenon.

TABLE II—Percentage of National Totals by EPA Region

EPA Region	Capital cost of industrial water treatment*	Total industrial capital cost	Annual cost of industrial water treatment	Value added by manufacturer†
	(Percent)	(Percent)	(Percent)	(Percent)
I	5.5	5.4	6.1	6.9
II	10.3	11.4	11.7	14.0
III	11.3	11.2	11.0	11.6
IV	14.0	15.1	14.1	12.5
V	24.9	30.5	24.6	30.1
VI	13.9	10.3	11.7	6.6
VII	3.5	3.4	4.3	4.7
VIII	2.1	1.1	2.5	1.3
IX	8.4	8.4	6.3	9.5
X	6.0	2.8	7.3	2.7

* From *1973 Economics of Clean Water*
† *Statistical Abstract of the United States,* Department of Commerce, 1972.

Some of the potential applications which would be meaningful at national, regional, state, and local levels are discussed below.

- The specific areal requirements which include the number and size of plants affected as well as the required outlays can aid government officials in evaluating specific plant by plant implementation plans in light of overall requirements.
- The alternative water use scenarios produce information which can be used to determine overall and average cost savings possible through greater water recycling or process changes. These savings vary greatly in the various geographical areas due to differing hydraulic efficiencies.
- Local cost data could be used to validate the model outputs for a particular geographic area. Also, differing types of treatment can be modeled if they apply in a specific area.
- Relating model outputs to other economic indicators can support analysis of relative impacts in different geographic areas or industrial subcategories.
- Much industrial wastewater is currently not treated at all. In some parts of the country, publicly owned treatment facilities may provide some of the required treatment. The model outputs can help aid local planning of industrial loading of publicly owned treatment works.

Sample application

One of the requirements of the Economics of Clean Water is the analysis of relative impacts of industrial water pollution control costs in states and EPA regions. To develop this impact, the capital requirements for water pollution control were compared with total capital expenditures and annual costs of water pollution control were compared to value added by manufacturing.

The percentage of the national requirement in each state and region for both capital and annual industrial wastewater treatment costs are based primarily on both the size and type of industry found in the state. The geographically-determined capital cost factor also makes a difference. The range by state is from less than 0.1 percent to 9.0 percent of total capital requirements and from less than 0.1 percent to 8.9 percent of total annual costs. Regional values for the former range from 2.1 percent to 24.9 percent and from 2.5 percent to 24.6 percent for the latter. Table I contains the percentages by state and Table II contains them by EPA region.

The relative share of industrial capital invested in a state or region is based on the growth rate for the area, the type of industry (capital intensive or not), and the age of existing facilities. Also reflected in the total capital expenditures are those expenditures for pollution control. The relative share of value added is also based on the size and type of industry.

While no direct relationship necessarily exists, it would seem that those areas with a larger share of capital requirements for pollution control than of capital expenses in general might encounter a greater burden in diverting capital to the construction of pollution control facilities. Examples of areas with this characteristic are Regions VIII and X and the following states:

Alaska	New Mexico
Hawaii	Oregon
Idaho	South Dakota
Maine	Vermont
Montana	Washington

Other areas might encounter less of a burden in diverting capital to construction of pollution control facilities given the capital cost and overall level of investment. Regions II, IV, and V fall into this category as well as the following states:

Arizona	Michigan
Connecticut	New York
Indiana	North Carolina
Iowa	Ohio
Kentucky	Tennessee
Maryland	Virginia

A similar comparison can be made between annual costs of pollution control and value added by industry. In those areas with relatively higher annual pollution control costs than value added, there may be greater changes in wages, prices, and dividends than in other areas. Areas with relatively high annual pollution control in comparison to value added are Regions VI, VIII, and X and the following states:

Alaska	Montana
Hawaii	New Mexico
Idaho	Oregon
Louisiana	Washington
Maine	Wyoming

Those with relatively high value added in comparison with annual costs are Regions II, V, and IX and the following states:

Arizona	Maryland
California	Massachusetts
Connecticut	Michigan
Delaware	New York
Indiana	Ohio
Kentucky	Tennessee

While no detailed set of effects by State or region can be developed from the data presented in Tables I and II, it is clear that significant differences are present between pollution control capital requirements and capital expended and between annual pollution control costs and value added. In fact, the two sets of measures generally reinforce each other which strengthens the hypothesis that there will be a differential burden.

CONCLUSIONS

The information made available by the Industrial Cost Model is useful to the Federal government in its planning and evaluation roles. In addition, it has potential applications at regional, state, and local levels of government and in the private sector. These other potential applications would primarily fall into the realms of planning and management.

The data and manpower resources required to develop the Industrial model are generally not available to state or local government. Thus, the development of the system by the Federal government and its being made available to other users present smaller units of government with a tool otherwise unavailable to them.

The base-data-cluster concept—A cooperative metropolitan approach to computer utilization

by LEONARD STITELMAN

Wayne State University
Detroit, Michigan

BACKGROUND

Metropolitan areas across the nation are devoting more attention to the general lack of current and detailed information so necessary for increasingly complex regional and local decision-making by public agencies. In the metropolitan Detroit, Michigan, region several groups have been involved in the attempt to develop a coordinated but voluntary approach to computer utilization which could serve as a model for other multi-jurisdiction metropolitan regions. It is known as the *Base-Data-Cluster Information System.*

The four million citizens of the six-county Detroit region, encompassing 4,000 square miles, are served by nearly 400 local political jurisdictions. These include counties, townships, municipalities, villages, school districts, and several types of special districts. The development of a regional cooperative computer plan initially came from public and private leadership in Southeast Michigan through the Metropolitan Fund, Inc., a nonprofit research corporation, which on a regional scale is similar in purpose and practice to many national foundations. The Fund's objective is to accomplish, through research, the physical and social goals for a better metropolitan way of life. The leadership for the Fund is provided by a 65 member Board of Trustees which includes the principal leaders of the educational, industrial, commercial, labor, and governmental segments of the community. Working closely with the Metropolitan Fund is the Southeast Michigan Council of Governments (SEMCOG), which shares similar objectives.

Two basic goals undergird the approach which was recommended for the local governments in the Detroit region:

1. The improvement of local government services and functions;
2. The development of a comprehensive routinely updated regional information-decision making network.

INITIAL DEVELOPMENT

In 1967 the Metropolitan Fund's first study on automation was published. It contained basic data on the extent of computer use by local governments, analyzed the effectiveness and impact of existing and proposed computer systems, and developed recommendations for the orderly and efficient use of automated equipment based on the principle of voluntary intergovernmental cooperation.

The 1967 survey revealed that comparatively few local governments had installed computers, and that such systems were usually not fully and effectively employed. The major problems were the shortage of qualified personnel and the limited individual government applications which did not utilize the EDP equipment for the minimum rental time.

In 1968 the Metropolitan Fund, in cooperation with SEMCOG and the Michigan Municipal League, sponsored a unique series of four educational Computer Forums, each held in a different county in the Detroit region. Every government jurisdiction with a computer system made a presentation which described its applications. These Forums, with total attendance of over 400 public officials, served as an extremely effective platform for the exchange of EDP information among both existing and potential users.

THE FEASIBILITY STUDY

In 1968, the Southern Wayne County Mayors Association (SWCMA) requested that the Metropolitan Fund conduct a study on the feasibility of their 18 communities joining in a cooperative computer service effort. The SWCMA, a voluntary aid group, consists of municipalities and townships in southern and western Wayne County, near the city of Detroit. Their total population is 507,000, with a range of 1,200 to 78,700.

While several communities are older, established ones, there are many currently experiencing the rapid residential and commercial expansion so common to suburban areas of metropolitan regions around the country. The member cities and townships of the SWCMA are a natural geographical and political grouping. Only one city, Wyandotte, with a population of 45,000, had a computer system, and several of its employees played a key role in the feasibility stage.

It was decided to analyze in depth four activities performed in all communities. These functions would serve as a model for determining the economic feasibility of the cooperative concept. Those selected were utility billing and maintenance, payroll, voter registration and maintenance, and property tax billing and collection. Seventy on-site interviews were conducted, with over 200 employees involved in these fact-finding sessions.

The project data was developed in two basic parts. The first was the preparation of an Information Study for each community. This was a detailed systems analysis and evaluation of the four applications. For each application there was a work flow, cost, personnel, and statistical analysis, and a general evaluation with recommendations for improving existing procedures, without regard to potential computer systems.

With this arrangement, the data privacy of each community was protected, and its relative effectiveness, or lack of it, as revealed in the comparative statistics, was made known only to the elected officials and their top staff. Valuable basic data on operations and procedures was thus secured, and for some communities this was the first time that their activities had ever been subjected to systems analysis. This information also provided the very crucial basis for comparing ongoing operating costs with future service bureau bids.

The second stage of the research project involved the preparation of a General Report, which contained the overall analyses, evaluations, and recommendations. Titled *Regional Cooperative Computer Plan*, it included the following major conclusions and recommendations:

(a) The sharing of a central computer system to service a group of local governments on a multi-jurisdiction, multi-application basis is technically sound and economically feasible.
(b) A cooperative computer service would provide significant improvements over present methods of information handling, and would definitely assist the communities in their administrative, managerial, and service functions.
(c) A decision by each community to seek computer capabilities on an independent basis would result in a substantial waste of tax dollars. Equal but separate computer facilities for each community would increase costs considerably and provide less service.
(d) A shared computer need not lead to conflicts in the scheduling services.
(e) Essential elements of privacy of data which each community may desire can be achieved.
(f) Implementation should be considered on an incremental basis—a short-range phased conversion to EDP using an established computer center, with future planning for an installation controlled or operated by the local governments being serviced. It was anticipated that such a center would expand to new applications, and that additional cities would join the cooperative.

ALTERNATIVE COOPERATIVE SYSTEMS

Local governments in the Detroit region have a long record in the use of contracts to perform elements of their services and functions. The Michigan State Constitution (Article VII, Section 28) permits counties, townships, cities, districts, or any combination thereof to:

(a) Contract with one another or with the state for the joint administration of their respective functions or powers;
(b) Share their costs and responsibilities of functions and services with one another and with the state;
(c) Transfer functions or responsibilities to one another with the consent of each unit involved:
(d) Cooperate with one another and with state government; and
(e) Lend their credit to one another in connection with any authorized publicly-owned undertaking.

With this very liberal basis for cooperation, the report proposed the following alternative systems:

(1) *Nonprofit corporation or independent authority under the direct supervision of the participating communities.*

Advantages:

(a) Direct control by elected local officials who would participate equally in all policy decisions.
(b) Staff directly responsible to governmental units.

Disadvantages:

(a) High start-up and implementation costs of computer service center.
(b) Large number of communities necessary to secure optimum benefits of equipment.

(2) *Single government unit acting as a service center.*

Advantages:

(a) City operating service center can test programs and procedures internally before implementation in other units.

Disadvantages:

(a) Fear of other communities that the one performing as a service center might receive favored treatment.
(b) Privacy of data fears.
(c) Contract for services not as desirable as direct policy oversight.

(3) *Commercial service bureau.*

Advantages:

(a) Commonly utilized arrangement.
(b) Commercial service bureau has already made its major investments in equipment and personnel.
(c) High start-up and developmental expenditures by local governments would be largely avoided.

Disadvantages:

(a) Concern of governments regarding assurances that their needs will be given the highest priority.
(b) Possibility of canned all-purpose programs which are not designed for special needs of governments.

(4) *County level government.*

Advantages:

(a) Base of cooperation broadened to additional government units and applications.
(b) Encourage additional coordination in non-computer functions because of increased contact on computer applications of communities.

Disadvantage:

(a) Civil service structure often limits development of appropriate staff.

ALTERNATIVE COST ESTIMATES

1. *Continuation of existing procedures.*

Average government expenditures have generally been rising at a rate of five to ten percent annually. For purposes of illustrating the impact of this alternative, the most conservative projection of a five percent increase was chosen. Any government agency, performing the identical function with the identical staff year after year, will show a rise in expenditures. Other typical factors, such as pay increases, also raise costs, so the five percent is a very modest projection. Here is how the governments in the study would be affected for the four applications under analysis:

Year	Annual Cost	Increase
1970 (current data)	$2,500,000	—
1971 (projected)	2,625,000	$125,000
1972 (projected)	2,756,000	131,000
1973 (projected)	2,894,000	138,000
1974 (projected)	3,039,000	145,000
1975 (projected)	3,191,000	152,000
1976 (projected)	3,351,000	160,000
1977 (projected)	3,519,000	168,000

The projected budget increase is over one million dollars for just four applications on a five percent rise per year. It is important to note that future computer system costs must be compared with *future* manual system costs, and not with current budget expenditures. If, with a computer system, we can reduce the *rate of increase,* in addition to other non-cost benefits, a major breakthrough will be achieved.

This position of continuing with existing manual procedures offers no alternative to the spiraling increases in public expenditures.

2. *Independent approach to computer utilization.*

What would be the cost impact of the go-it-alone approach to data processing? Let us assume that only the twelve cities of over 10,000 population decide to utilize computers, and that the others are too small. Let us also assume that each city utilizes a computer service bureau and hires a basic systems staff.

Utilizing figures developed by the Municipal Information Technology Program of the University of Connecticut, the result was a per capita cost of $2.82.

3. *Cooperative approach to computer utilization.*

Using the twelve cities in the example above, let us assume a decision to use a computer cooperatively. Data was developed for batch and real time processing, and comparative expenditures are as follows:

Alternative	Annual Cost	Per Capita Cost
Independent (service bureau)	$1,348,000	$2.82
Cooperative (batch processing)	$ 596,000	$1.25
Cooperative (real time processing)	$ 766,000	$1.60

The question of cost of a cooperative computer service center is the crucial determinant as to whether such a center is established. We must not fall into the trap of comparing today's non-computer, manual operations budget against tomorrow's computer based budget. Today's costs do not remain the same for tomorrow. Budgets are increasing at a rate of five to 10 percent a year with no corresponding increase in services or operations. Automated systems offer the alternative of reducing the rate of increase.

THE BASE-DATA-CLUSTER CONCEPT

The major significance of this study did not lie in the conclusion that cooperation is beneficial to local governments. Although this fact cannot be emphasized too strongly, and the report documents the savings factors, the basic contribution of this study is to provide a *workable* prototype for the creation of a comprehensive regional information-decision-making system with automatic updating features. This prototype, the base data-cluster concept, utilizes two essential elements:

1. *Tie the information base to the most elemental origination point available.* The production of a water bill, a paycheck, or a property tax bill inherently includes much of the data which eventually finds its way into even the most sophisticated MIS. In addition, an information system is meaningless unless it is timely—updated quickly, regularly, and economically. Such a network plugged into the source of data origination in the city results in an automatic by-product. Thus,

by concentrating our energies on the "nuts and bolts" of providing basic government services, there comes the assurance of the political support necessary for a functioning regional system. This is the key to the realistic implementation of such a system—it will be approved more for its internal service value than for its comprehensive information system.

2. *Build upon groupings or clusters of local governments utilizing subregional computer service centers on a cooperative basis.* Without the cluster concept the base-data principle is unworkable. If each community adopted the "go-it-alone" philosophy, we would have hundreds of computer inputs to a regional computer center, an overwhelming organizational problem. The other extreme, the huge central computer with terminals in every community, was rejected as politically impractical. Thus, we have the cluster approach in which voluntary multi-jurisdictions EDP service centers are formed based primarily on geography and political compatibility. While we are keenly aware that this approach presents some problems, the alternatives (go-it-alone and one-giant computer) would be financially disastrous and/or unworkable.

This concept of computer utilization in our metropolitan regions, while easy to describe, is very difficult to implement. Positive inputs from other government levels would facilitate the objectives stated above. The Federal government, particularly through the grant mechanism, should be actively encouraging cooperative arrangements, since such contacts also serve to break down other barriers among local jurisdictions; cooperating on a computer inevitably leads to cooperation on other mutual problems. Washington has funded the development of vertical municipal information systems, which, viewed in the context of metropolitan goals, is a negative influence. However, it should be noted that several years ago, an HEW grant stimulated an outstanding example of voluntarism and clustering. It resulted in a computer service center operated by the Oakland County Intermediate School District, north of Detroit. Securing the voluntary approval of 28 local school districts, the Intermediate District has developed a wide range of administrative and education computer applications. It has provided considerable encouragement to our thesis that voluntary cooperation is practical.

State governments must also play a larger role. Some have assisted this cooperation concept through the passage of legislation to standardize requirements, such as uniform accounting records. More of this is needed, not to stifle local initiative, but to bring sound procedures to all local jurisdictions, thereby enhancing their ability to cooperate voluntarily.

Of great relevance for the future of the base-data-cluster approach is the increasing importance of regional councils of governments (COGs). When several subregional clusters are established in a metropolitan area, it is possible that the COG might serve as the link which brings together the centers' data and other federal, state, regional, and private sector information. The COG could also serve as the stimulus for the approval of computer clusters by member communities.

CONCLUSION

The strength of the base-data-cluster theory is that it does not attempt to superimpose preconceived ideas on the development of the clusters. They must be built on the strength of geography, politics, and personalities, in whatever combination is workable. This will not result in a neatly defined complex of centers; machine interface will pose a serious problem.

Nevertheless, the positive indirect results must be emphasized. Cooperation with respect to automation can lead to other cooperative activities by local governments. A demonstration of mutual benefits involving computers cannot help but reduce distrust and misunderstanding on other issues which must be handled through voluntary cooperation.

As the population of our metropolitan regions increases, resulting in a corresponding pressure for government services, and the information requirements become more critical, computer utilization becomes as inevitable as the conversion from the pen to the typewriter. Basically, the issue is not whether—but how. The "how" will influence metropolitan politics in many ways for many years. And unless local governments do join together in cooperative arrangements, they will not receive the benefits and advantages which today's computer technology offers.

SOURCES

Stitelman, L., *Automation in Government: A Computer Survey of the Detroit Metropolitan Region,* November 1967: Metropolitan Fund, Inc., Detroit, Michigan.

Stitelman, L. and Dennis Little (editors), *Proceedings of Southeast Michigan Detroit Region Computer Forums,* July 1969; Metropolitan Fund, Inc., Detroit, Michigan.

Stitelman, L., *Regional Cooperative Computer Plan,* February 1970: Metropolitan Fund, Inc., Detroit, Michigan.

Efficiency in generalized pipeline networks*

by C. V. RAMAMOORTHY and H. F. LI

University of California
Berkeley, California

INTRODUCTION

A common architecture of most of today's super machines revolves around parallel or pipeline processing. Typical examples of such machines are the CDC STAR-100, TI-ASC, PEPE, IBM 360/91, 360/195, and CDC 6600, 7600, etc.[1-4] They all have distinct pipeline processing capabilities, either in the form of internally pipelined arithmetic functional units or in the form of a pipeline of special purpose functional units. The principal idea behind pipelining is to create as much overlap as possible in the operations of the different facilities, for example, memory fetch unit, decoding units, adders, and multipliers. Concurrency of different operations increases the system utilization. As an important consequence of concurrency, the execution speed of most jobs are accelerated considerably as is evidenced in systems like 360/195 and TI-ASC. Ideally, in a pipelined machine, instead of obtaining one output per major cycle from the system, a rate of one output per minor cycle may be achievable. A typical linear pipeline is as drawn in Figure 1(a).

The functioning of a pipeline can be illustrated as that of a water-pipe. In the ideal case where all the facilities have the same speed, the analogous water-pipe has a uniform cross-section (Figure 1(b)). On the other hand, when some of the facilities are slower, the corresponding cross-sections in the water-pipe are narrower as drawn in Figure 1(c). From the figures, it is intuitively obvious that the facilities of a pipeline should be chosen so that their speeds are about the same, in order to achieve maximum efficiency.

In this paper, the pipelining concept is generalized. In the generalization, instead of having only one pipeline in the system, a network of pipelines crossing at certain points (facilities) are installed. In this respect, another interesting analogous example can be given. A network of pipelines can be viewed as a traffic network. The facilities shared by different pipelines (or paths) are the junctions in the traffic. With this picture in mind, several immediate observations can be obtained. Everyone realizes the importance of traffic lights in a busy traffic. They have the responsibility of regulating the various roads (analogous to pipelines) so that maximum throughputs are attainable. Then the roads are said to be well-utilized. The same reasoning holds for a pipeline network. The traffic light at the entrance of a pipeline is actually the sequencer while a traffic light inside the network is a decentralized control containing the necessary buffers. This remarkable similarity between a pipeline network and traffic network is extremely useful for visualizing and designing a modern, advanced computer system. As a practical example, the TI-ASC computer has a multi-pipeline feature as shown in Figure 2.

In this paper, an efficiency measure is formulated to justify the design of shared resource pipeline network. The design of the network and the sequence control will be discussed. Particular attention is paid to the overhead associated with optimal sequencing. In order to reduce the overhead, local optimization in the static case is proposed. Results of this study reveal several directions for future investigation, and they will also be discussed.

EFFICIENCY ANALYSIS OF PIPELINES OF VARIOUS CONFIGURATIONS

The linear (straight-line) pipeline system

In Reference 6, the author proposes to view the efficiency of a linear pipeline system to be the ratio of the total space-time span of the jobs to the total space-time span of the facilities (processing stations) concerned. A simple efficiency measure was suggested under the assumption that the execution times of the stations are the same so that no buffering or particular bottleneck exists inside the pipeline (the slowest facility of the linear pipeline is called the bottleneck).

To further generalize this useful measure of efficiency, one can naturally consider also the cost and speed associated with each facility. This is especially important because in practice various facilities have different speeds (or cycle times). As is almost universally accepted, if a designer is willing to pay more, he may find a faster facility to be installed in the system at a higher cost. Hence a unit of idle time in some fast facility should be counted more costly than

* Research sponsored by the National Science Foundation, Grant GJ-35839, and the U.S. Army Research Office—Durham, Grant DA-ARO D-31-124-73-G157.

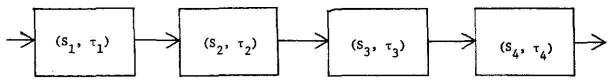


Figure 1(a)—A typical linear pipeline

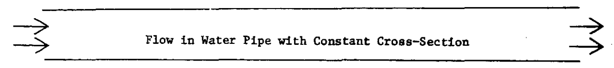


Figure 1(b)—Analogous Waterpipe when $\tau_1 = \tau_2 = \tau_3 = \tau_4$

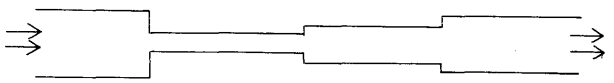


Figure 1(c)—Analogous Water Pipe when $\tau_1 \neq \tau_2 \neq \tau_3 \neq \tau_4$

that of a slower one. This implies that an appropriate weighing factor should be attached to the space-time span of each facility in analyzing the efficiency of the system.

At the same time, in the case of a linear pipeline, it can be shown that additional internal buffering is quite unnecessary even when the facilities have different execution times. Instead, a buffer of suitable size located at the entrance of the linear pipeline may serve very well in regulating the pump-in rate of the data or tasks to be processed. This idea can best be illustrated with an example. In Figure 3(a), the system has large enough buffers (intermediate storages between facilities) to hold all intermediate information to be transferred, and in Figure 3(b), buffering is not inserted anywhere inside the pipeline, except at the entrance. It can be observed directly from the figures that in both cases, the system has the same response time, throughput rate and efficiency. (Notice that all operations are assumed to be synchronous).

FACT: In a linear pipeline system, the throughput rate is determined by the speed of the slowest facility in the pipeline (the bottleneck); whereas the response time is given by the sum of the time spent in all of the facilities in the pipeline.

From the example in Figure 3(b), the throughput rate depends entirely on the speed (τ_j) of facility S_j and hence the maximum throughput rate of the system is $1/\tau_j$. From this, one can observe that removal of comparatively slow bottlenecks will be of vital importance in pipeline designs.

We are now ready to define the efficiency of a linear pipeline system according to space, time, cost and speed.

Let τ_j = speed of the bottleneck

τ_i = speed of the ith facility in the pipeline

α_i = weight attached to the space-time span of the ith facility as determined by the cost and speed of the facility, for example, cost X speed product.

L = number of tasks pumped into the pipeline in a certain period of time. For maximum efficiency, it will be assumed that the tasks are pumped in continuously, that is, the buffer at the entrance of the pipeline is never empty.

n = number of facilities in the pipeline.

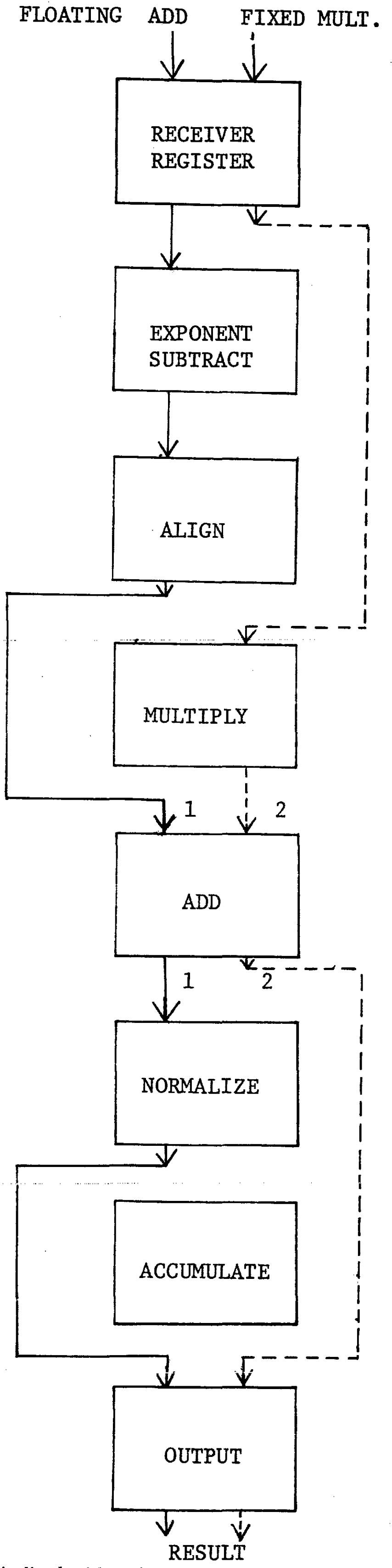


Figure 2—Pipelined arithmetic unit in TIASC—An example

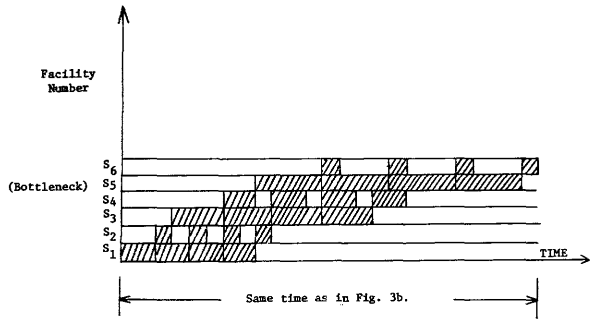


Figure 3(a)—Gantt chart for a pipeline with infinite intermediate buffering

Then

$$\text{Efficiency} = \frac{\text{total weighted space-time span of } L \text{ tasks}}{\text{total weighted space-time span of } n \text{ facilities}}$$

i.e.

$$\eta = \frac{L(\sum^{n} \alpha_i \tau_i)}{\sum^{n} \alpha_i (\sum^{n} \tau_i + (L-1)\tau_j)} \qquad (1)$$

The individual terms in the above measure, as indicated in Figure 4, are easily comprehensible because they include important parameters such as cost, speed, and space-time. In the ideal situation where all facilities have the same speed, the equation simplifies into

$$\eta = \frac{L}{n+(L-1)}$$

so that when L approaches infinity, the efficiency approaches unity. In all other cases, as L approaches infinity, the efficiency approaches

$$\frac{\sum^{n} \alpha_1 \tau_1}{(\sum^{n} \alpha_i)\tau_j} < 1.$$

Bottleneck removal

Following the analysis in the previous section, a natural question arises: if an identical facility is connected in parallel across the bottleneck S_j, what kind of performance improvement may be achieved? In what way will the efficiency be affected?

For the simple linear pipeline, the situation is not difficult to analyze. Intuitively, when two identical units operate in parallel to serve input requests, the speed of the combined unit may become halved (that is, $\tau_j/2$), provided that the input rate is suitably regulated. As a result of this observation, the combined parallel unit in the new system will remain to be the bottleneck under the following condition.

Lemma

The two parallel facilities treated as a combined unit remains to be the bottleneck of the system if and only if for all

$$i \neq j,\ 2\tau_i \leq \tau_j \ (\text{or } \tau_i \leq \tau_j/2).$$

Proof

This assertion can be visualized using a Gantt chart as drawn in Figure 5. Referring to the figure,

(1) for $i \leq j$, $2\tau_i + \tau = \tau_j + \tau \Rightarrow \tau_j \geq 2\tau_i$

and

(2) for $k \geq j$, $\tau_1 + \tau_j + d + \tau_k \leq 2\tau_j + d \Rightarrow \tau_j \geq 2\tau_k$ Q.E.D.

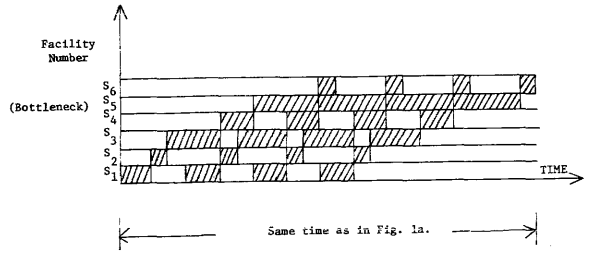


Figure 3(b)—Gantt of same system without intermediate buffers

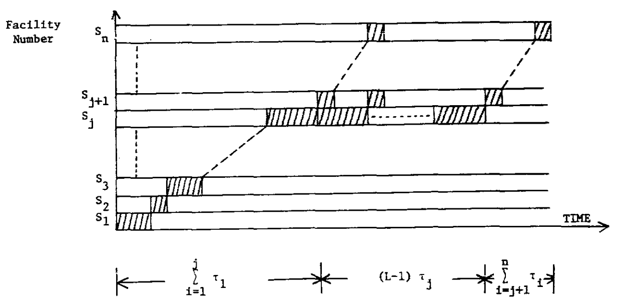


Figure 4—Efficiency of a linear pipeline

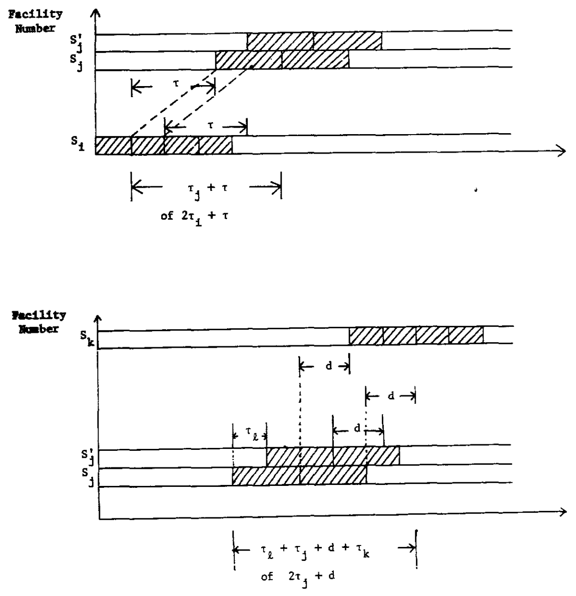
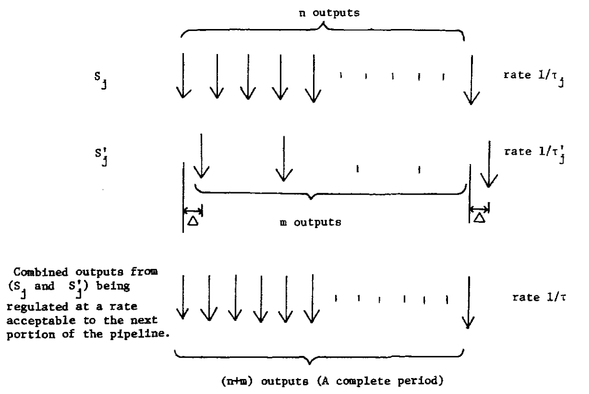


Figure 5—Two identical parallel stations remain bottleneck



n outputs
S_j
rate $1/\tau_j$
S_j'
rate $1/\tau_j'$
Δ
m outputs
Combined outputs from (S_j and S_j') being regulated at a rate acceptable to the next portion of the pipeline.
rate $1/\tau$
(n+m) outputs (A complete period)

Figure 6—Buffer control for 2 parallel units

If the previous condition is satisfied, the throughput rate of the system can be regulated to $(2/\tau_j)$—twice the throughput obtained before the addition of the facility S_j'. A design option becomes apparent; it may be desirable to insert a buffering control between the combined facility (S_j and S_j') and the next facility S_{j+1} so that the outputs from the combined unit can be buffered and transmitted to the next portion of the pipeline at a regular speed. This may also be accomplished externally at the entrance of the pipeline, but the decentralized control inside the system will serve a very useful purpose later when more complicated pipeline configurations are involved.

The efficiency of the new system when the combined unit remains to be the bottleneck is given by:

$$\eta' = \begin{cases} \dfrac{L(\sum^n \alpha_i\tau_i)}{(\sum^n \alpha_i+\alpha_j)\ (\tau_l + \sum\limits_{i\neq j}^{n} \tau_i + L/2\tau_j)} & \text{if } L \text{ is even} \\[2ex] \dfrac{L(\sum^n \alpha_i\tau_i)}{(\sum^n \alpha_i+\alpha_j)\left(\sum\limits_{i\neq j}^{n} \tau_i + \left(\left[\dfrac{L}{2}\right]+1\right)\tau_j\right)} & \text{if } L \text{ is odd} \end{cases}$$

When L becomes very big,

$$\eta' \approx \frac{\sum^n \alpha_i\tau_i}{(\sum^n \alpha_i+\alpha_j)(\tau_j/2)} \neq 2\eta \text{ as defined in eq. (1)}$$

On the other hand, when the condition in the Lemma is not satisfied, a new bottleneck is created. By applying the same analysis as discussed earlier, the efficiency of the new system becomes:

$$\eta' = \frac{L(\sum^n \alpha_i\tau_i)}{(\sum^n \alpha_i+\alpha_j)(\sum^n \tau_i + (L-1)\tau_k)} \qquad (2)$$

where τ_k = the speed of the new bottleneck.

A few observations can be made concerning the previous analysis. First, efficiency is not always improved. If there exists a new bottleneck whose speed $\tau_k \approx \tau_j$, then equation (2) indicates that $\eta' < \eta$ while the throughput rate is not much improved, (being $1/\tau_k$). Second, if the new bottleneck has a speed $\tau_k \ll \tau_j/2$, then the new facility S_j' need not operate as fast as S_j. This initiates our next investigation.

Parallel facilities with different execution speeds

The problem to be solved here is to choose a suitable facility to be connected across a bottleneck in order to fulfill a certain maximum throughput requirement at a cost-effective manner. This is a common problem faced in many practical designs. For example, suppose the bottleneck identified is a multiplier, and the designer has the option of introducing one or more multipliers to operate in parallel with the original one. Then the multipliers may be chosen from a set of available candidates, some microprogrammed, and some hardwired (fast but expensive).

If $1/\tau$ is the throughput rate desired for the design, and the present bottleneck has a rate of $1/\tau_j$, intuitively the speed requirement (τ_j') on the parallel facility S_j' to be connected in parallel with S_j is that $1/\tau = 1/\tau_j + 1/\tau_j'$. Yet there lies behind the bushes a problem not to be overlooked. Because the two facilities S_j and S_j' have different speeds, their outputs will be out of phase for most of the time. In fact, their outputs may collide or may be asynchronous to the extent that the maximum combined output is not periodic without proper control. This situation can be described as drawn in Figure 6.

Referring to Figure 6, the buffer control after the combined unit (S_j and S_j') is responsible for receiving the outputs from the latter and send them to the next portion of the pipeline at an acceptable rate. Whether this can be done at a constant period depends on finding the integer solution (n, m) for the

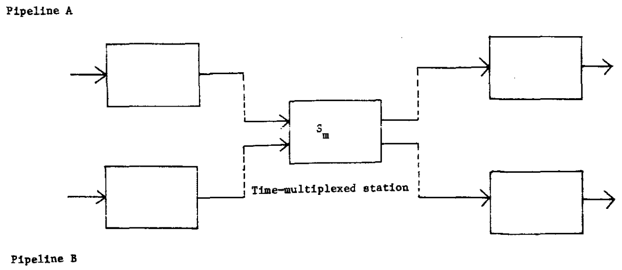


Figure 7—A shared resource pipeline

equation $n\tau_j + m\tau_j' = (n+m)\tau$, given the values of τ, τ_j, and τ_j'. If no such integer solution exists (as may occur), either the combined output rate τ or the speed τ_j' of S_j' must be adjusted. Fortunately it can be shown that the size of the additional buffer to regulate this output rate need not be very big. In fact, a single stage buffer is capable of doing the job as proved in the following.

Proof

Assume at time $t=0$, the facility S_j receives the input task flowing in. For any integer k, then (see Figure 6):

$$\left[\frac{k\tau}{\tau_j}\right] = m_0,$$

$$\left[\frac{k\tau}{\tau_j'}\right] + 1 = m_1 = \text{maximum number of outputs generated from } S_j' \text{ in the interval.}$$

But, $m_0 + m_1 \leq k$ if none of the bracketed values is an exact integer (which is true within the $n\tau$ interval used to describe a complete period of the combined output). Hence a single stage buffer control may accomplish the task of regulating the output as no catch-up is possible for the outputs from S_j and S_j'. Q.E.D.

In order that this scheme of regulated output is implemented properly, the different tasks or data to flow into the pipe must carry with themselves identification tags; otherwise, when the output order differs from the input order, erroneous results are obtained. Hence, this decentralized control involves some additional overhead, and its implementation should be considered very carefully.

Shared resource pipeline

In a linear pipeline where the facilities have different speeds, efficiency is rather low as explained earlier. To improve utilization (increase the overlap of operations and decrease the idle time of existing expensive units), shared resource pipeline design is a solution. The term 'shared resource pipeline' refers to two (or multiple) pipelines sharing an expensive (but fast) facility. For example, suppose a facility S_m in a linear pipeline A has a speed of $\tau_m \leq \tau/2$, where τ is the maximum speed of excitation of pipeline A (A pipeline is said to be excited when a task is admitted into the pipeline). Then the expensive facility S_m can be shared (via multiplexing) by some other pipeline B which also requires this facility as shown in Figure 7. This may be viewed as the *dual* of the previous case where more than one parallel facility are serving one task stream.

Under this design configuration, some local control must be installed at the entrance of the shared facility (compared to that with the buffer control in the previous section) so as to resolve conflict and utilize the resource efficiently. In this respect, two alternatives could be classified: (1) the tasks could be scheduled before entering the pipeline so that no conflict will result, or (2) a local control is set up at the shared facility as illustrated. The design strategies and sequencing disciplines will be explored in detail in the next section.

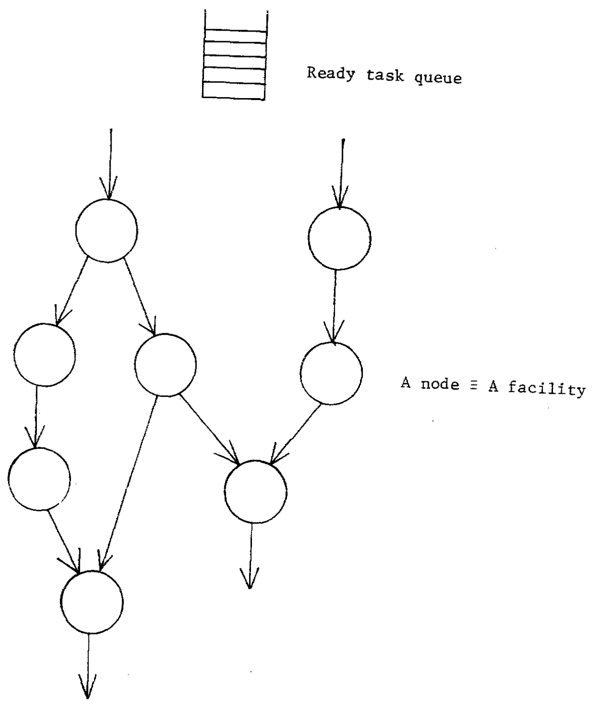


Figure 8—A generalized pipeline system

Coming back to the system drawn in Figure 7, the efficiency of the combined system, assuming that S_m is not a bottleneck, is given by:

$$\eta' \approx \frac{L\left(\sum^{n} \alpha_i\tau_i\right) + L'\left(\sum^{n'} \alpha_i'\tau_i'\right)}{\sum^{n} \alpha_i\left(\sum^{n} \tau_i + (L-1)\tau_j\right) + \sum_{i\neq m}^{n'} \alpha_i'\left(\sum^{n'} \tau_i' + (L'-1)\tau_j'\right)} > \eta$$

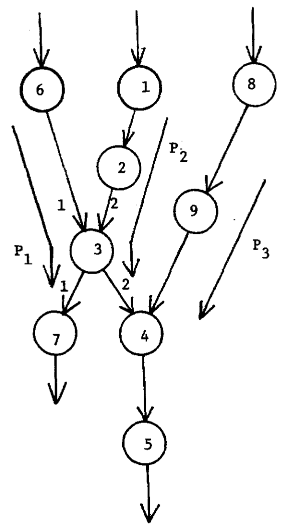
for the case where S_m is duplicated.

However, if S_m becomes the bottleneck of the combined system, it may be disastrous (especially when the maximum throughput rate then cannot be achieved). In such a case, proper scheduling becomes extremely important for resolving conflicts. Perhaps, the shared facility S_m should also be duplicated eventually.

GENERALIZED PIPELINE NETWORK

General discussion

Subject to the previous type of analysis, a system consisting of several sequential and parallel facilities can be interconnected in almost any possible configuration, as long as the throughput and efficiency analyses suggest it is the



$$T_s = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & t_{23} \\ 0 & t_{32} & t_{33} \end{pmatrix}$$

$$R^3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Figure 9(a)—An example system with collision matrix T_s and the relation R^i.

optimal way to do so. Going one step beyond, each of these facilities could be a giant functional unit, for example, a minicomputer in a network of computers. Hence the problem that will be dealt with in this section may be of paramount importance in future advances in computer architecture.[16]

Before going further, it is appropriate at this point to define some terms that will be often used. A set of data or instructions allowed to enter a pipeline to be processed as a group will be called a task. This definition agrees with the conventional definition of the term task (which is often stated as a part of a program which once initiated can be executed to completion without further input). On the other hand, when a task tries to enter a facility which is busy serving some other task, a collision occurs, and the only remedy is to buffer the task at the entrance of the facility or the pipeline. Figure 8 illustrates an example of a generalized pipeline system. From it, one can realize that such a generalized system can be modelled as a directed graph. A node in the graph represents a facility and an arc represents a possible transition. Hence each feasible path in the graph represents a pipeline and parallel processing is intrinsically introduced into the model—a hybrid mode of operation results.

The problems associated with such a design can be summarized into the following:

(1) Blocking and unblocking paths at the correct moment (it will be assumed that each task carries with it an identification tag).
(2) Resolving or avoiding collisions by proper buffering control as described before.
(3) Increasing the throughput of the system by proper sequencing.
(4) Developing some strategies for controlling loops in the pipeline graph.
(5) Improving the execution time of some particular program graphs by proper pre-scheduling. However, this topic will be out of the scope of this paper.

Model

In this section, the design of a generalized pipeline system or network will be discussed. For the time being, it will be assumed that there is no maximally strongly connected subgraphs in the pipeline graph (that is, there is no loop in the system). This assumption will be removed after the basic concepts have been presented.

From the pipeline graph, the different paths of the system are known. Each path in the graph represents a pipeline which can operate in parallel with the other pipelines in the system, sharing particular facilities wherever appropriate. For the convenience of discussion, each feasible path is labelled P_i (ith pipeline). From these paths and the speeds of the facilities, a simplified collision matrix for the system can be constructed. The matrix T_s has its entry t_{ij} representing the time after the initiation of a task in P_i such that another task can be admitted into P_j without any collision. Obviously when P_i and P_j do not share any facilities, the entries $t_{ij}=t_{ji}=0$. With this matrix, the tasks to be distributed among the various pipelines may be sequenced such that no collision occurs and hence no intermediate storage within the pipeline system will be necessary. Hence this design option avoids collision at the expense of efficiency.

Even in this simple case, one complication has to be resolved. A graph may contain some nodes with multiple (or single) entry and multiple-exit arcs. In some cases, some of the exit arcs are not legal for some of the entry arcs of the node simply because those permutations are not desirable in the original design. Therefore, for such a node k, a relation matrix must be set up to map these arcs correctly. The R^k matrix so defined contains entries

$$r_{ij} = \begin{cases} 1 & \text{if inward arc } i \text{ can go to outward arc } j \\ 0 & \text{otherwise.} \end{cases}$$

A simple example to illustrate these ideas is shown in Figure 9(a). As a practical counterpart, assuming each facility takes the same time, the matrices for the TIASC example given in Figure 2 are constructed as shown in Figure 9(b).

In addition, in order to control the flow of a task from an inward arc to a correct outward arc when there are more than one choice (which is indicated by more than a single 1 in the row corresponding to that inward arc in the R^k matrix),

$$T_s = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix} \qquad R^{ADD} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Figure 9(b)—The matrices for the TI-ASC example in Figure 2, assuming each facility has the same speed of 1 unit of time

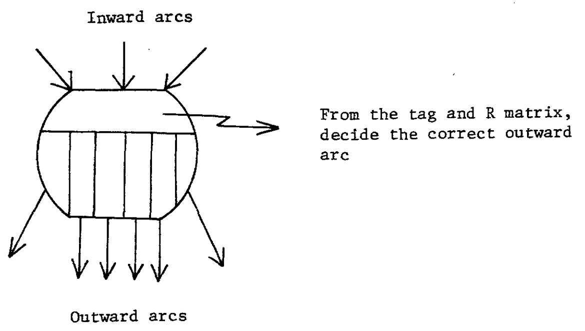


Figure 10—A multiple-entry multiple exit node

a tag must be associated with the task to indicate which particular path to follow as illustrated in Figure 10.

Under these simplified conditions, the control mechanisms of the pipeline network can be easily implemented. Globally, to avoid collision anywhere inside the pipeline system, the designer only has to set up counters at the entrance of the system, one for each path. For the example in Figure 9(a), three counters are necessary. Counter 1 will be initiated to up-count from zero when P_1 is excited, and similarly for counters 2 and 3. Counter i will stop counting when it has reached the ceiling equal to the maximum entry in the ith row of the simplified collision matrix (T_s). This signifies that from that time on, any task to be admitted into the system will not create any collision in P_i. Hence, a state of the pipeline system can be conveniently defined using the contents of these counters. If the content of counter i is represented by C_i, the state$\triangleq(C_1, C_2, \ldots, C_i, \ldots, C_n)$ where n represents the number of pipelines in the system.

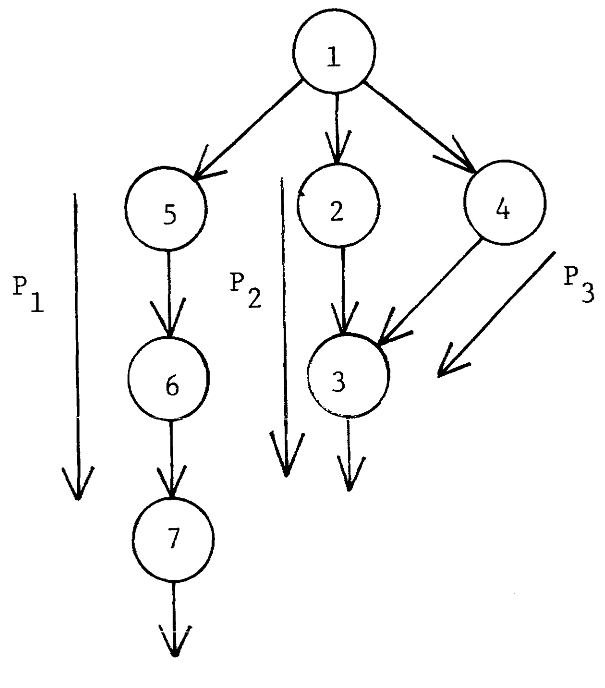


Figure 11(a)—An example system

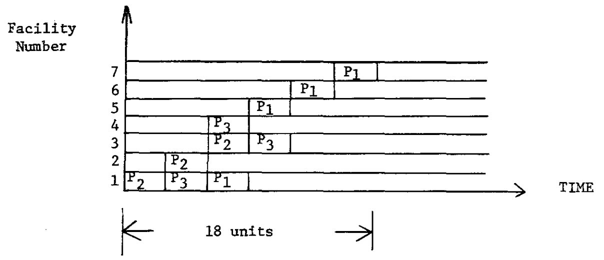


Figure 11(b)—FRFS schedule

A Simple Admission Strategy:

If a task is waiting to be admitted to pipeline P_i, it will be allowed to enter if and only if for all j, $C_j \geq t_{ji}$. This merely means that it has no conflict with any previous tasks being processed in the system.

Evaluation:

A few characteristics of the previous scheme must be evaluated. First, the scheduling strategy used is identical to the FRFS (First Ready First Served) scheme commonly used in time sharing systems. However, FRFS is by no means optimal. A simple counter example is set up in Figure 11a where the system has three paths P_1, P_2, and P_3. For simplicity, assume that all facilities have the same speed, say 3 units of time. At time $t=0$, P_2 is excited, and at $t=3$, a task for P_3 is ready. If P_3 is excited at $t=3$ without considering the arrival of a task for P_1 at $t=4$, the resulting schedule is drawn in Figure 11(b). However, if P_1 is allowed to be excited before P_3 (thus between $3 \leq t \leq 4$, the first facility is idle), the resulting schedule is as drawn in Figure 11(c). From these schedules, it can be concluded that FRFS is not optimal; for the example, the FRFS schedule takes 2 units of time more, and efficiency is diminished.

This example exposes the important effects of sequencing in a general pipeline system, similar to other multiprocessor systems. But unfortunately, simple, non-enumerative optimal algorithms for the general case have not been found. Improved sequencing disciplines will be the subject of the remaining part of this paper.

A second observation is that the simplified collision matrix only describes the earliest times after which a corresponding

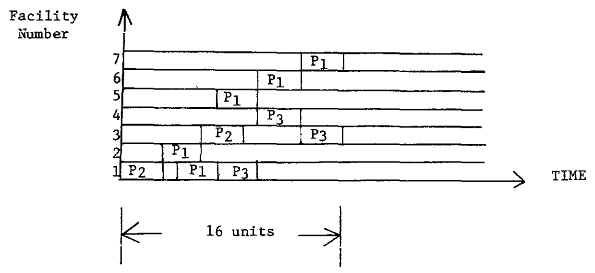


Figure 11(c)—Better schedule

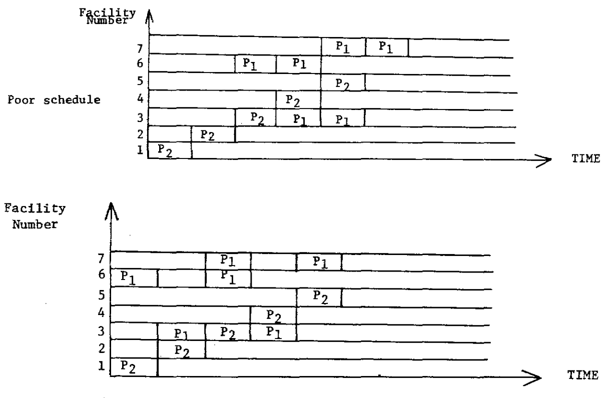


Figure 12—Schedules for system in Figure 9(a)

excitation of a pipeline will not produce any collision inside. It does not describe the earliest possible times to initiate a task in which collisions are avoided. This may seem to be a strange assertion but can be illustrated using the example graph in Figure 9. In this system, assume all facilities have the same execution speed, say 1 unit of time, and P_1 and P_2 collide at facility S_i. If at time $t=0^-$, P_2 is excited, but at time $t=0^+$, two tasks request for admission into P_1. From the simplified collision matrix, $t_{21}=2$ so that P_1 will not be excited until $t=2$. Consequently a poor schedule is obtained as compared to the optimal one as in Figure 12.

In order to obtain the optimal schedule, more sophisticated global control must be installed at the entrance of the system. In particular, each entry in the collision matrix will not be single-valued; instead, each entry is described by a m-tuple of the form of $t_{ij}=((t_{ij}^{11}, t_{ij}^{12}), \ldots, (t_{ij}^{m1}, t_{ij}^{m2}))$. For example, $(t_{ij}^{11}, t_{ij}^{12})$ represents the first interval of time between which P_j can be excited without colliding with P_i (assuming that at time $t=0$, P_i is excited). The implementation becomes more complicated because more hardware will be needed to keep track of these time intervals. Nonetheless, the previous idea of counter control may be similarly exploited and the details will not be given here. Notice that in the latter implementation, multiple collisions between two pipelines P_i and P_j are also tolerable (a multiple collision occurs when two paths cross at more than one node as indicated in Figure 13).

An alternative implementation for situations where multiple collisions occur should be mentioned. Control buffers can be inserted at shared facilities to accommodate unexpected multiple collisions. However, then the beauty of global control may be spoiled while throughput is not guaranteed to improve (the additional hardware and expensive logic—including duplication of buffers at every shared facility—should not be overlooked). For purposes of this paper, such trade-off criteria will not be discussed.

Sequencing to increase efficiency and throughput

A major design objective in pipeline systems, quite contrary to that of parallel processing (which is to minimize the execution time of particular programs), is to increase the system utilization and throughput as much as possible. The example in the previous section reveals that FRFS is not optimal in many cases and proper sequencing must be adopted in order to reach the goal. The implementation of a sequencing strategy can be treated as an additional facility to be placed at the entrance of the pipeline system; thus the overhead of sequencing is overlapped with the other operations in the pipelines. As a result, throughput may be improved while response time is at most slightly affected, depending on the complexity of the sequencing facility.

However, unfortunately, the problems of optimal scheduling facing a general pipeline system are even more difficult to solve than that facing ordinary multiprocessor systems. The reasons are twofold. First, there are many different pipelines in the system interconnected in any fashion, unlike the identical processors in most parallel processing systems. Second, since there are shared facilities in the system, collisions must be avoided (even in the case where buffers reside inside the system, excess collisions are intolerable because buffers must have finite sizes).

There are two kinds of sequencing environments, namely, dynamic and static. In this paper, the term dynamic sequencing refers to the condition under which the arrival of the tasks will be random and their exact paths are not known. Obviously, in such a situation, nothing much could be done without creating too much overhead; and heuristics such as FRFS or modified FRFS (mixed with some dynamic priority scheme) may be adopted. Hence, dynamic sequencing will not be of particular interest here.

In the static case, it is assumed that a set of ready tasks are waiting at the input queue to be admitted into the system. So the sequencer is responsible for sequencing them in a proper order to be admitted by looking into a subset of the tasks in the queue. Sequencing is of utmost importance in a

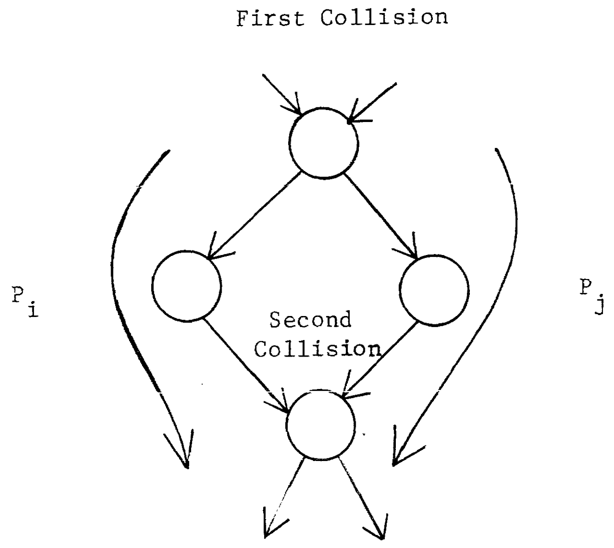


Figure 13—A multiple collision situation

general pipeline system because by developing a proper strategy, the efficiency and throughput of the system may be greatly improved. To further emphasize this concept, suppose the pipeline system has four different paths (hence four parallel pipelines). By statically analyzing the superiority of some sequences of a certain fixed length L, a priority scheme can be established. Then the sequencer only has to look ahead into the queue for L tasks and sequence them according to the result of the static analysis done before. This analysis is a fixed cost of the system—no run time overhead is involved. The result of the analysis may be stored in an associative table. This scheme will be explained in more detail later.

Simple as this approach may appear, there does not exist any non-exhaustive algorithm which can generate the optimal priority listing for the L tasks being considered (supposing the sequencer looks ahead L tasks at a time). The problem for this local optimization will be formulated as follows:

Let S be an ordered sequence $= (u_1, \ldots, u_k)$ representing the indices of the paths that the k tasks to be sequenced will travel.

Then, $t_{u_1 \cdots u_i} =$ the minimum elapsed time before path P_{u_i} can be excited if the pipelines are excited in the order $(u_1, \ldots, u_i)$.

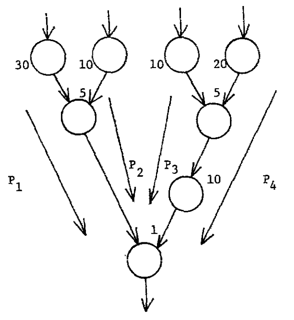
$$= \underset{\forall j \neq i}{\mathrm{Max}} \{t_{u_1 \cdots u_j} + t_{u_j u_i}\}$$

and $E(S) =$ the execution time of the sequence S

$$= \mathrm{Max}\,\{E(u_1, \ldots, u_{i-1}); t_{u_1 \cdots u_i} + E(u_i)\}$$

Observe that this recursive equation takes into account all possible cases, including those where some tasks are subsumed by others (a task is said to be subsumed by some other tasks if its execution phase is completely overlapped with the latter. This relation of subsumption is not transitive).

The optimal sequence for the set of tasks to be scheduled corresponds to the sequence whose execution time $E(S)$ is minimum. By asserting this condition, obviously both efficiency and throughput rate are improved. A branch-and-bound solution for the situation where no subsumption of tasks exists can be found in Reference 14. But in general if some tasks are subsumed by others (which frequently occur), an almost exhaustive approach seems inevitable. Another difficulty in solving this problem lies in the fact that $t_{u_1 \cdots u_i}$ for all possible permutations of the set S cannot be obtained easily. It involves solving the longest path problem between all pairs of nodes (obeying each sequence ordering concerned) of an associated graph containing positive cycles. The nodes in this associated graph represent the pipelines to be excited (observe that a pipeline can be excited more than once, in which case, a node will be duplicated), plus a sink node as shown in Figure 14. An arc leading from node i to node j has a value of t_{ij} representing the minimum time elapsed between the excitation of P_i and that of P_j. An arc from a node to the sink has a value equal to the execution time of the pipeline P_i. In Figure 14, it has been assumed that all four pipelines are to be excited.



$$T_s = \begin{pmatrix} 30 & 25 & 11 & 1 \\ 0 & 10 & 0 & 0 \\ 0 & 11 & 10 & 0 \\ 1 & 21 & 15 & 20 \end{pmatrix}$$

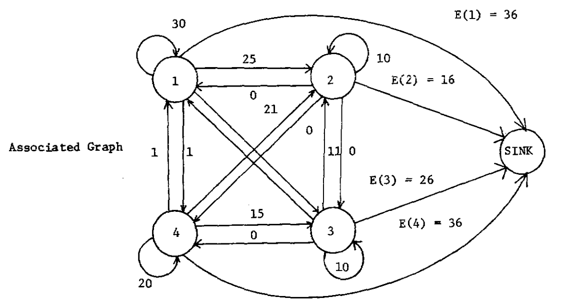


Figure 14—The associated graph of an example system

The presence of cycles in the associated graph inhibits the use of some iterative algorithms (for example, Floyd's algorithm for shortest paths) for finding the longest paths between all pairs of nodes. Instead, the individual subgraphs (containing no loops) corresponding to certain orderings are extracted and analyzed for the longest paths. Tragically, because of this, some useful sequence dominance criteria cannot be established.

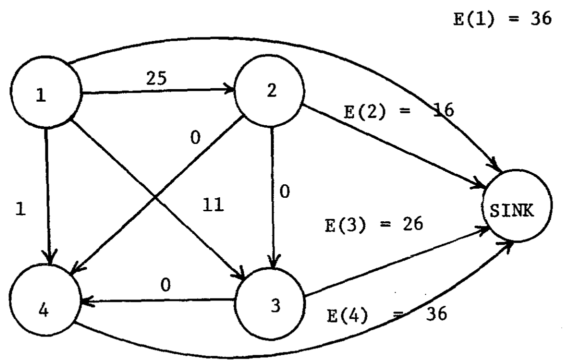
But when the pipeline system has very few pipelines, for example, 4 pipelines, it is still desirable to apply static sequencing to produce some locally optimal sequences. With a look-ahead set of 4 tasks, the objective can be accomplished. The procedure of optimization can be illustrated as follows.

Step 1:

For the set of tasks being looked ahead, say u_1, u_2, u_3, and u_4 belonging to $\{1, 2, 3, 4\}$ corresponding to each path in the system in Figure 14, choose an ordering, for example (u_1, u_2, u_3, u_4). Extract from the associated graph those permissible transitions. Find the longest path which leads from u_1 to the sink. If the longest path does not go through all the intermediate nodes in the isolated subgraph, conclude that $E(u_1, u_2, u_3, u_4) \leq E(u_1, u_3, u_2, u_4)$. This is a sequence dominance criterion which holds only for the case of four tasks (because there are only two permutations for the two intermediate nodes). In the example, the result is shown in Figure 15.

Step 2:

Repeat Step 1 for all ordered sequences of the set



E(1,2,3,4) = 61 $\subseteq$ E(1,2,3,4) by the dominance rule

Figure 15(a)—Extracted subgraph for the ordering (1, 2, 3, 4)

$$L = \begin{bmatrix} - & 48 & 61 & 61 \\ \textcircled{37} & - & 42 & \textcircled{37} \\ 48 & 42 & - & 48 \\ 57 & 52 & 42 & - \end{bmatrix} = \text{Longest path matrix.}$$

an optimal ordering for {1,2,3,4} is (2,3,4,1)

Figure 15(b)—The longest path matrix

$\{u_1, u_2, u_3, u_4\}$ unless the particular sequence has been dominated. A longest path matrix can be constructed. The P_{ij} entry in this L matrix contains the minimum execution time for the sequence starting with u_i and ending with u_j. For the example, the L matrix is drawn also in Figure 15.

Step 3:

Choose the entry in the L matrix containing the minimum value compared to the rest. This corresponds to the locally optimal sequence for the set of tasks considered. For the example, the sequence (2, 3, 4, 1) or (2, 3, 1, 4) can be chosen.

If local optimization is performed for all different sets of tasks containing 4 elements, a table could be set up for the optimal sequences for these sets. For example, the sets may be {1, 1, 1, 2}, {1, 2, 2, 3} etc., and hence there are $4^4-4=252$ entries in the table. An associative search could be adopted to sequence the four tasks in the look-ahead set.

The size of the associative map can be further reduced by recognizing the different permutations of the same combination of tasks, for example, {1, 2, 3, 1} is the same as {1, 2, 1, 3} as well as many others. If only distinct combinations of the tasks are considered by proper addressing, the general expression which yields the size of the associative map needed for a n-pipe system can be represented by $g(n)$ where

$$g(n) = f(n, n) - n$$

and

$$f(m, x) = \sum_{i=0}^{m} f(m-i, x-1)$$

for $m \leq n$, $x \leq n$.

The initial conditions for the above recursive equation are:

$$f(p, 1) = f(0, q) = 1$$

for all p and $q \leq n$.

From this equation, it can be found that in a four-pipeline system, the number of entries in the map is 31. This means a considerable reduction compared to 252. In the case of a five-pipeline system, the corresponding figure is 121, which is still a tolerable figure in practice.

The major difficulty in deriving the value of $E(S)$ is because the associated graph contains positive cycles so that finding the longest paths between some nodes and the sink could not be done iteratively. Consequently, no simple dominance criteria could be found to eliminate some inferior permutations of the look-ahead set.

In any case, this type of local optimization can be applied to any kind of generalized pipeline system because the overhead is negligibly small—an associative search and then the appropriate ordering. Throughput is improved by proper sequencing as illustrated and response time in the worst case is only slightly degraded.

Further application

The sequencing strategy just described can also be extended and applied to tasks which have different execution time requirements on the facilities, as occurring in the case of computer network. Obviously, in order that the overhead pays off, the execution times of the tasks must be long compared to the overhead of sequencing. This is easily justified if we have separate jobs to be processed through a network of computers. The latter can be modelled as a generalized pipeline system whose facilities have variable speeds. Thus local optimization must be done dynamically with different sets of tasks being looked ahead.

In this situation, the sequencer has to analyze the collision matrix and derive the optimal sequence according to the same technique described in the previous section. When the overhead is intolerable, then simple heuristics could be used. A simple rule might be as follows: from the set of tasks, choose the task to enter P_i such that the smallest entry in the ith row of the collision matrix is also minimum among the set. This rule merely asserts that then it will be more probable for some other task to be initiated at the earliest possible moment after the excitation of P_i. In any case, it is strongly felt that sequencing heuristics in this area have to be further studied in future because they will be of significant usage in computer networks.

Finally, if a loop exists in the system, a feasible strategy is to break the loop so that the task is sent back to the ready queue to be sequenced again. This may improve the system utilization, because proper sequencing can be applied, though the execution speed of that particular loop may be delayed. If delay for some task is undesirable, the task may be given a high dynamic priority so that the sequencer will admit it as soon as no collision will occur.

CONCLUSION

In the preceding sections, the concept of pipeline has been generalized to a network of facilities. Both feasibility and suitability of such a pipeline design have been discussed, based on an efficiency measure which considers all important system parameters such as cost, speed, space-time span and desirable goal of throughput. For a complicated shared resource pipeline, its efficiency can be deduced from the efficiencies of the various paths (pipes) of the system, employing the analysis presented in this paper. But it should be pointed out that in such a case, each individual efficiency must be appropriatelv weighted according to its usage frequency statistically.

A lot of non-trivial problems have been discovered and they have to be solved before such a generalized system can be well-utilized. Among these problems are the buffering control and sequencing strategy. The different ideas presented in this paper by no means represent the end of the investigation. In fact, they may well mark the beginning. More future efforts should be spent to develop better sequence dominance criteria or heuristics in particular systems for sequencing so as to reduce the overhead involved, especially in the case where it has to be performed dynamically (in a computer network). The static look-ahead sequencer described in this paper may be used in most systems, because it involves very little overhead.

There are some advantages of a generalized pipeline system which must be mentioned. A generalized pipeline system (or network) can be reconfigured dynamically subject to the change in system environment. Furthermore, the reliability and availability of the system may be better because of its network nature. These latter aspects, together with the prescheduling of program graphs and dynamic priority assignment for particular tasks, are well worth looking into in future.

REFERENCES

1. Hintz, R. G., and D. P. Tate, "Control Data STAR-100 Processor Design," *Compcon* 1972.
2. Watson, W. J., "The TI-ASC—A Highly Modular and Flexible Super Computer Architecture," *AFIPS, FJCC, 1972*, pp. 221-230.
3. Anderson, D. W., F. J. Sparacio, and R. M. Tomasulo, "The IBM System/360 Model 91: Machine Philosophy and Instruction Handling," *IBM Journal of Res. and Dev.*, Vol. 11, No. 1, Jan. 1967, pp. 8-24.
4. Thornton, J., "Parallel operation in the Control Data 6600," *AFIPS Proc.*, *FJCC*, Vol. 26, pp. 33-40, 1964.
5. Chen, T. C., "Distributed Intelligence for User-Oriented Computings," *AFIPS Proc.*, FJCC, 1972, pp. 1049-1056.
6. Chen, T. C., "Parallelism, Pipelining, and Computer Efficiency," *Computer Design*, Jan. 1971, pp. 69-74.
7. Flynn, M., and A. Podvin, "Shared Resource Multiprocessing," *Computer*, March/April 1972, pp. 20-28.
8. Flynn, M., "Some Computer Organizations and Their Effectiveness," *IEEETC*, Vol. c-21, No. 9, Sept. 1972, pp. 948-960.
9. Graham, R., "The parallel and the pipeline computers," *Datamation*, Vol. 16, April 1970, pp. 68-71.
10. Baer, J. L., "A survey of some theoretical aspects of multiprocessing," *Computing Survey*, Vol. 5, No. 1, March 1973.
11. Cotton, L. W., "Maximum rate pipeline systems," *Proceedings SJCC*, 1969, pp. 581-586.
12. Conway, R., *Theory of Scheduling*, Addison Wesley, Reading Mass. 1967.
13. Reddi, S. S., and C. V. Ramamoorthy, "Some Aspects of Flow-Shop Sequencing Problems," *Sixth Princeton Conference on Information Sciences and Systems*, Princeton, N.J. March 1972.
14. Reddi, S. S. and C. V. Ramamoorthy, *Sequencing Strategies in Pipeline Computer Systems*, Technical Report No. 154, Electronic Research Center, the University of Texas at Austin, August 1972.
15. Davidson, E., *The Design and Control of Pipeline Function Generators*, Stanford Report, 1972.
16. Ramamoorthy, C. V. and K. H. Kim, "Pipelining: The Generalized Concept and Sequencing Strategies," *Proc. NCC*, 1974.

An approach to the design of highly reliable and fail-safe digital systems*

by HENRY Y. H. CHUANG

University of Pittsburgh
Pittsburgh, Pennsylvania

and

SANTANU DAS

North Electric Company
Delaware, Ohio

INTRODUCTION

Recent progress in electronic technology has resulted in extensive use of electronic digital or logical systems in many complex control processes where weight or size as well as reliability and safety is important. Typical examples are, to name a few, nuclear reactor control, missile guidance systems, "fly-by-wire" systems in aircraft or spaceship control, and electronic telephone switching systems (ESS). In such applications, if the system is not well protected, some failure could result in a catastrophic accident or unacceptable loss in terms of lives and resources. For the system to be safe and reliable, not only the computers or digital systems in the control loop have to be reliable, but also the sub-systems which take part in the data acquisition or control should be fail-safe and reliable. In order to lessen the chance of system breakdown and unsafe failures in applications of this nature, many protective methods have been proposed and studied. Generally, the methods can be classified as: (1) fault-masking approach,[1-9] where the emphasis is on achieving higher reliability by using redundancy to mask failures, and (2) fail-safe approach[10-14] where attention is directed to the design of systems in such a way that they take a predetermined safe state whenever failures occur.

Although reliability and fail-safety are both vital in many applications, a design technique which can have the merits of both the fault-masking approach and the fail-safe approach has been strikingly lacking. By properly combining the two redundancy methods (in particular, the NMR and the N-fail-safe logic), we have obtained a design approach for realizing reliable and fail-safe systems which can also have several significant advantages. These include excellent degradability, good testability, design flexibility and simplicity, as well as real-time error indication. In this article the authors expand and generalize the concepts presented in their previous short paper,[15] and try to throw light on its many advantages.

* This work was supported by National Science Foundation under Grant GJ-35569.

THE NEW FAULT-RESTORATION METHOD

The new fault-restoration scheme is shown in Figure 1. Here the individual logic blocks have two levels of failure.[15,16] In the first level, which we call the "safe output" level, the output is not correct but its value is something different from normal operating values of the output of the logic blocks. In the second level of failure, the output is neither "correct" nor "safe", and the value it takes up is one of the normal operating values (but which is not what it is supposed to have). The restorer's function is to give correct output as long as there are more copies having correct outputs than those having incorrect output, no matter how many copies have safe output. In case there are as many copies with correct output as with incorrect output (again no matter how many other copies have safe output), then the restorer output would be a safe value. The restorer function for a three-copy scheme is shown in Table I for elucidating the principle of this scheme. Here each of the three-copies, named A, B, and C, can have three distinct output states, R (right or correct), W (wrong or incorrect) and S (safe). It is assumed that in each logic block, the probability of getting the safe output S is higher than that of having an incorrect output. In other words, more internal failures are required to result in an incorrect output than a safe output. This safe output S can be used to indicate the necessity of outside intervention and, depending on the application, provisions could be made for either manual or automatic corrective measures (say replacement of the faulty unit). Thus unlike the majority voting[1,4] or other fault-restoration schemes, this scheme will have two levels of failure. The result is that the system would have a

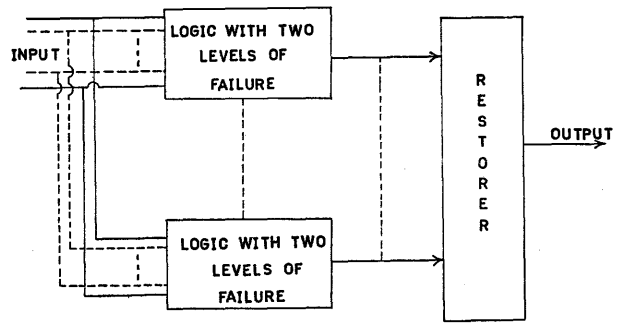

INPUT
LOGIC WITH TWO LEVELS OF FAILURE
LOGIC WITH TWO LEVELS OF FAILURE
RESTORER
OUTPUT

Figure 1—Fault-restoration using multiple copies of identical logic blocks (each logic block having two levels of failure)

TABLE I—Restorer Function of a Three-Copy Scheme

A	B	C	Output
S	S	R	R
S	R	R	R
R	R	R	R
W	R	R	R
S	S	S	S
W	S	R	S
W	W	S	W
W	S	S	W
W	W	W	W
W	W	R	W

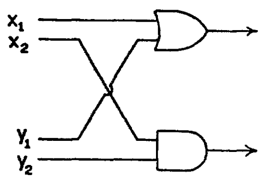
high correct output reliability and still higher *safe output reliability,* and so reliability and fail-safety can be achieved simultaneously.

To realize such a system, ideally each individual copy in Figure 1 would be made of such logic primitives as will produce safe output S in case of failure, and the failures in the primitives would tend to manifest themselves at the outputs of the copies by driving them to S. Most of the common logical components do not exhibit such failure characteristics. In addition, since most available logical components are two-valued, we have to resort to some sort of coding to build the system we have envisioned, using available electronic technology. One practical solution is to use the N-fail-safe logic[13,14] to realize the individual copies of logic. In the next section a short overview of N-fail-safe logic would be given.

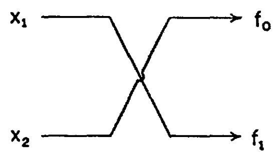
N-FAIL-SAFE LOGIC

The truth table for minimal information loss[13,14] N-fail-safe NOT, OR, and AND are given in Table II; and Figure 2(a-c) shows their double-rail realizations. By interchanging the two output terminals of OR and AND, we have NOR or NAND respectively. These are called N-fail-safe primitives. In this figure, the variable x is coded by (x_1, x_2), y by (y_1, y_2), and f by (f_0, f_1). Logical 0 and 1 are represented by (0, 1) and (1, 0) respectively, and N is represented by either (0, 0) or (1, 1).* One way to synthesize an N-fail-safe logic circuit of a given Boolean function is to first synthesize a non-fail-safe circuit and then replace each gate by its corresponding N-fail-safe primitive. Obviously the circuits synthesized by this replacement method contain at most twice as many components as needed by their non-fail-safe versions.

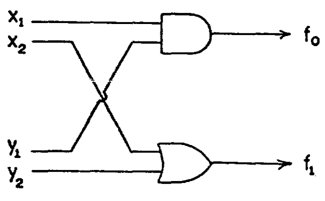
It is to be noted that, in such OR and AND primitives, one or both of the gates failing in the same direction (i.e., both fail to 1 or both to 0) will result in an N output even if there is an N input, provided it is also due to a fault in the same direction. Therefore, in an N-fail-safe logic system, single fault or multiple faults all in the same direction will always result in N output, while multiple faults not all in the same direction may result in an erroneous 0 or 1 output. Thus, if the N-fail-safe logic is built of *ideal asymmetrical* components then it will function perfectly. In our method, the N-fail-safe logic is applied in such a way that even if non-asymmetrical components are used the system would still have very high reliability and fail-safety.



(a) N-fail-safe OR.



(b) N-fail-safe NOT.



(c) N-fail-safe AND.

Figure 2—N-Fail-safe primitives

TABLE II—Truth Table for Minimum Information Loss N-Fail-Safe NOT, OR, and AND

Variables		NOT	OR	AND
x	y	$\bar{x}$	x+y	x·y
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1
0	N	1	N	0
N	0	N	N	0
1	N	0	1	N
N	1	N	1	N
N	N	N	N	N

* Takaoka assumed only 0 → 1 failure and therefore N is (1, 1) in his case.

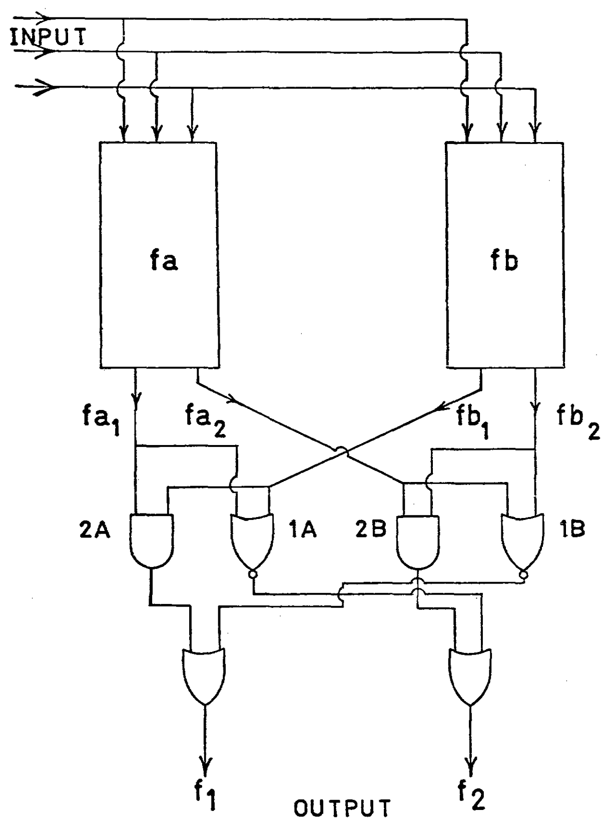


Figure 3a—Fault-restoration using two copies of N-fail-safe logic

fb_1fb_2 \ fa_1fa_2	(N) 00	01	(N) 11	10
(N) 00	N	01	N	10
01	01	01	01	N
(N) 11	N	01	N	10
10	10	N	10	10

(b) Map for Restorer

fb_1fb_2 \ fa_1fa_2	00	01	11	10
00	11	01	00	10
01	01	01	01	00
11	00	01	11	10
10	10	00	10	10

(c) Map for the Restorer Circuit o:

Figure 3b and c

REALIZATION OF FAULT-RESTORATION

The implementation of our fault-restoration method using the double-rail N-fail-safe logic is best elucidated by using a simple example. Figure 3a shows a possible scheme for restoration using two copies of N-fail-safe logic. The network f_a and f_b are identical copies of N-fail-safe logic realizing a function f. The restorer produces output according to the combinations shown in Table III. In the table, columns A and B list outputs of the two copies and the "output" column gives the restorer output. N (which corresponds to safe state "S") denotes either (0, 0) or (1, 1), R the correct output, and W the incorrect output.

TABLE III—Restorer Output For Two-Copy System

A	B	Output
R	R	R
N	R	R
R	N	R
N	N	N
N	W	W
W	N	W
W	W	W
R	W	N
W	R	N

Figure 3b shows the map for the restorer function. For realization, the N entries in the map are replaced arbitrarily by either (0, 0) or (1, 1).

Figure 3c gives the specific assignment used for the restorer shown in Figure 3a. It is to be noted that this restorer is also fail-safe.

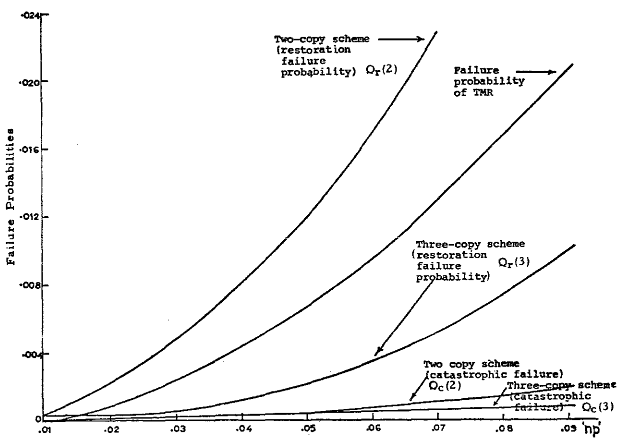


Figure 4—Failure probability curves versus "np"

FAILURE PROBABILITIES

Unlike the other redundancy schemes, the scheme described here has two levels of failure: (1) restoration failure (when the output is not correct), and (2) catastrophic failure (when the output is neither correct nor N). Since each N-fail-safe primitive has at most two gates, if a Boolean function needs n gates in its realization, its N-fail-safe version will need at most $2n$ gates. In the following we first derive the failure probability formulas for our two-copy system of Figure 3 and then generalize them.

(1) *Probability of N Output*

Let $p=s+t$ be the failure probability of a gate, where s and t denote the probabilities of failing to 0 and to 1 respectively. The system can fail to correct faults and produce N output in two different ways: (1) one (or more) gate in each copy fails in the same direction (all fail to 0 or all fail to 1), and as a result both the copies give output N, which is passed on to the final output by the restorer, and (2) two (or more) gates fail in opposite directions in one of the copies, thus giving rise to a W value at the output of that copy. In the latter case, even if the other copy is working correctly, the restorer output will be N.

The probability of case 1 can be approximated by that of two gates failing in two separate copies, and is given by*

$$Q_1=[2np(1-p)^{2n-1}]^2 \simeq 4n^2p^2$$

The probability of case 2 can be approximated by that of two gates failing in opposite directions in a copy, which is

$$Q_2=2\binom{2n}{2}\times 2st\times(1-p)^{2n-2} \simeq 8n^2st$$

The maximum value of st is $p^2/4$. Thus

$$Q_2 \simeq 8n^2p^2/4=2n^2p^2$$

* This product np is small in practice (usually $\ll 1$). So, higher order terms are negligible.

The total probability of N output in a two-copy system is thus given by

$$Q_N(2)=Q_1+Q_2=4n^2p^2+2n^2p^2=6n^2p^2$$

(2) *Probability of W Output*

An output W will occur if, because of some faults, one of the copies gives W output while the other copy has an output N. (We can ignore the case when both copies have W output as that probability is very small.) If it is assumed that any two gates failing in different directions in one copy give rise to a W output, while a single gate failure in another copy gives rise to N at its output, then the probability of W output for a two-copy system is

$$Q_W(2)=2\times 2np\times\binom{2n}{2}\times 2st(1-p)^{2n-2}(1-p)^{2n-1}$$

$$\simeq 4np\times 4n^2\times p^2/4=4n^3p^3$$

(3) *Restoration and Catastrophic Failure Probabilities*

The restoration failure occurs when the output is either N or W, while the catastrophic failure occurs when the output is W. Therefore the restoration failure probability $Q_r(2)$ is

$$Q_r(2)=Q_N(2)+Q_W(2)=6n^2p^2+4n^3p^3 \simeq 6n^2p^2$$

and the catastrophic failure probability $Q_c(2)$ is simply $Q_W(2)=4n^3p^3$.

These are the worst case failure probabilities, as the aforementioned assumption may not result in these faults and as it has been assumed that each N-fail-safe primitive has two gates which actually is not the case for the NOT primitive. Moreover, the actual value of st will be far less than $p^2/4$ because most electronic components have asymmetric failure characteristics (i.e., $s \neq t$). When ideal asymmetrical elements are used,

$$Q_r(2)=Q_1=4n^2p^2, \quad \text{and} \quad Q_c(2)=0.$$

Detailed comparison between our scheme and the other popular redundancy schemes are given in Reference 16.

(4) *Generalization*

As mentioned earlier, the restoration strategy can be extended to systems using any number of copies, odd or even. Let P_s, P_w, and P_r be the probabilities of a copy to have "safe", "wrong", and "right" output respectively and if x is the number of copies, then it can be shown[16] that the probability of N output for an x-copy system is:

$$Q_N(x)=\sum_{k=0}^{[x/2]}\frac{x!}{(x-2k)!k!k!}(P_s)^{x-2k}(P_w)^k(P_r)^k,$$

and that of w output for an x-copy system is:

$$Q_w(x)=\sum_{i=0}^{x-1}\sum_{j=0}^{[(x-i-1)/2]}\frac{x!}{i!j!(x-i-j)!}(P_s)^i(P_r)^j(P_w)^{x-i-j}$$

Where $[U]$ denotes the largest integer $\leq U$. As before, the probability of w output is really the catastrophic failure probability, while the sum of the probabilities of w output and N output is the restoration failure probability. Hence,

$$Q_r(x) = Q_N(x) + Q_w(x)$$

and

$$Q_c(x) = Q_w(x)$$

When double-rail N-fail-safe logic is used for realization, good estimations of P_s, P_r, and P_w, as explained earlier, are given by:

$$P_r \approx (1-p)^{2n} \approx 1$$

$$P_s \approx (1-p)^{2n-1} 2np \approx 2np$$

$$P_w \approx (1-p)^{2n-2} \binom{2n}{2} \frac{2p^2}{4} \approx n^2p^2$$

The failure probabilities for two-copy and three-copy systems plotted along with that of the TMR (three-copy majority voting) are shown in Figure 4.

ADVANTAGES

The new scheme of fault-restoration, we are advocating, is superior in performance to most of the existing redundancy schemes. Some of the important advantages are as follows.

(1) *High Reliability and Fail-Safety*

As shown in Figure 4, the (restoration) reliability of our scheme is comparable to that of TMR. But, our catastrophic failure probability is very low. Thus, our scheme can also provide good fail-safe protection which is not available in other fault-restoration schemes. A more comprehensive reliability comparison with other fault-tolerant design methods is given in Reference 16.

(2) *Excellent Degradability*

Our scheme has also degradability unmatched by any other existing redundancy method. As for instance, switching from triplex system (using three copies) to a duplex system is very easy as one can always build the logic in such a way that shutting off the power of one of the copies would force its output to N. Similar switching from duplex to simplex or for that matter from any number of copies to any smaller number of copies is possible without changing the restorer. This is so because our restorer for higher number of copies always logically covers that for lower number of copies. Among all other redundancy schemes, TMR (or NMR) is the only scheme that is degradable. But even for TMR (or NMR) this sort of degradability is absent as it cannot switch from triplex to duplex, although it can switch to simplex from triplex. To do even that one has to bypass the voter which could be unwieldy.

(3) *Trade-off Between Restoration and Catastrophic Failures*

One can make trade-offs between restoration failure and catastrophic failure probabilities by altering the restorer logic or the restoration strategy, keeping the number of logic copies unchanged. As for instance, Table IV shows the different behaviors of two restorers along with the probabilities for various possible output combinations of a three-copy system. From this table, we get for restorer 1:

Table IV—A Three-copy System with Two different Restorers

A	B	C	OUTPUT OF RESTORER 1	OUTPUT OF RESTORER 2	APPROXIMATE PROBABILITIES
N	N	R	R	R	$3(2np)^2 = 12n^2p^2$
N	R	R	R	R	$3(2np) = 6np$
R	R	R	R	R	$(1-np)^3 \simeq 1$
W	R	R	R	R	$3n^2p^2$
N	N	N	N	N	$(2np)^3 = 8n^3p^3$
W	N	R	W	N	$2np \times n^2p^2 = 2n^3p^3$
W	R	N	N	N	$2np \times n^2p^2 = 2n^3p^3$
N	W	R	W	N	$2np \times n^2p^2 = 2n^3p^3$
R	W	N	N	N	$2np \times n^2p^2 = 2n^3p^3$
N	R	W	R	N	$2np \times n^2p^2 = 2n^3p^3$
R	N	W	R	N	$2np \times n^2p^2 = 2n^3p^3$
W	W	N	W	W	$3\,(n^2p^2)^2(2np) = 6n^5p^5$
W	N	N	W	W	$3\,(n^2p^2)(2np)^2 = 12n^4p^4$
W	W	W	W	W	$(n^2p^2)^3 = n^6p^6$
W	W	R	W	W	$3\,(n^2p^2)^2 = 3n^4p^4$

$$\text{restoration failure probability} \simeq 16\, n^3p^3$$
$$\text{catastrophic failure probability} \simeq 4\, n^3p^3,$$

and for restorer 2:

$$\text{restoration failure probability} \simeq 20\, n^3p^3$$
$$\text{catastrophic failure probability} \simeq 15\, n^4p^4.$$

Thus, we see that by using restorer 1 instead of restorer 2 we can decrease the restoration failure at the cost of increasing the catastrophic failure. Restorer 2 corresponds to the restoration strategy based on which the generalization of failure probability analysis is done.

One of the realizations of restorer 1 is shown in Figure 5. Of course, one can always realize the restorer circuits in the same way as we did for the 2-copy restorer of Figure 3.

(4) *Amenability to Hybrid Redundancy*

In situations where a very high mission life is to be achieved spares can be used, as in the case of TMR, to obtain dynamic or hybrid redundancy. As the safe output S indicates failures in the network, it can be conveniently used to initiate the necessary switching of spares. Because in our system any number of copies, odd or even, can be used, the spare switching circuits are simpler.[16]

(5) *Amenability to Simple MOS IC Realization*

It is well-known[19,20] that it is comparatively easy to realize monotonic functions using current MOS IC technology. Since functions f_0 and f_1 in an N-fail-safe logical system are both monotonic, our fault-restoration method has an edge

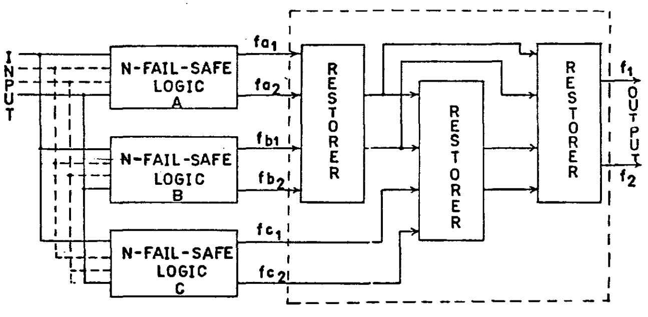


Figure 5—A three-copy system (here the restorer is composed of three 2-copy restorers as shown in Figure 3(a))

over other methods from realization point of view. Further, it should be pointed out that even if one uses N-fail-safe logic, it is not necessary that one has to build it by using N-fail-safe primitives. As has been shown in Reference 13, it is also possible to obtain f_0, f_1 of the N-fail-safe logic from direct realization. In such a case, two separate LSI (or MSI) chips can be used to realize f_0 and f_1. It is very likely that one of these LSI chips would malfunction at a time and not both at the same time. Thus, when a copy fails, its output will most likely be N and not W.

(6) *Good Testability*

Protective redundancies might mask detectability. This is the case for most fault-masking methods including the quadded logic[7] and interwoven logic.[8] Some non-protective redundancies such as those for obviating hazards could also hinder fault detectability. However, the N-fail-safe logic, with its monotonic component functions, has inherently good testability. It has been reported[17,18,21] that in a monotonic logic network not only all faults are detectable but also the minimal detection cover can be easily obtained. Further, it can be shown[16] that in an N-fail-safe logic network even the redundancies for avoiding hazards cannot mask its fault detectability.

CONCLUSIONS

A new design method for the realization of highly reliable and fail-safe digital systems has been described here. The method properly combines the fault-masking and the fail-safe approaches, and thus possesses the merits of both. It also has significant advantages, such as the excellent degradability and two-level protection, which are not available in existing redundancy methods.

A major disadvantage of this method could be the high cost of realization because a copy of N-fail-safe logic *might* require twice as many gates and connections to realize as would that of conventional logic. However, it should be noted that the N-fail-safe component functions are always monotonic and monotonic functions generally have simpler IC realizations. Consequently, this demerit might not be so bad at all.

REFERENCES

1. Von Neuman, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," *Annals of Mathematical Studies*, No. 34, pp. 43-98, Princeton University Press, Princeton, New Jersey, 1956.
2. Moore, E. F., and C. E. Shannon, "Reliable Circuits Using Less Reliable Relays," *Journal of the Franklin Institute*, Vol. 262, pp. 191-208, September 1965; pp. 281-297, October 1965.
3. Hamming, R. W., "Error Detecting and Error Correcting Codes," *Bell System Technical Journal*, Vol. 29, pp. 147-160, 1950.
4. Pierce, W. H., *Failure-Tolerant Computer Design*, Academic Press, New York, 1965.
5. Knox-Seith, J. K., *A Redundancy Technique for Improving the Reliability of Digital Systems*, Stanford Electronics Laboratory, TR No. 4816-1, December 1963.
6. Teoste, R., "Digital Circuit Redundancy," *IEEE Trans. on Reliability*, Vol. R-13, pp. 42-61, June 1964.
7. Tryon, J. G., "Quadded Logic," in *Redundancy Techniques for Computing Systems*, Wilcox and Mann, eds., pp. 205-228, Spartan Books, Washington, D. C., 1962.
8. Pierce, W. H., "Interwoven Logic," *Journal of Franklin Institute*, Vol. 277, pp. 55-85, 1964.
9. Finkelstein, H. A., *An Investigation into the Extension of Redundancy Techniques*, Co-ordinated Sciences Laboratory, University of Illinois, Report R-455, February 1970.
10. Mine, H., and Y. Koga, "Basic Properties and a Construction Method for Fail-Safe Logical Systems," *IEEE Trans. on Electronic Computers*, Vol. EC-16, No. 3, pp. 282-289, June 1967.
11. Hirayama, H., Y. Urano, and T. Watanabe, "Synthesis of Fail-Safe Logic Systems," *Electronics and Communication in Japan*, Vol. 52-C, No. 1, 1969.
12. Urano, Y., *System Theory—On the Synthesis of Highly Reliable Logical Systems*, Ph.D. dissertation, Waseda University, Tokyo, Japan, November 1969 (in Japanese).
13. Takaoka, T., *Algebraic Theory of Automata and Its Application to Fail-Safe Systems*, Ph.D. dissertation, Dept. of Applied Math. and Physics, Kyoto University, Kyoto, Japan, December 1970.
14. Takaoka, T., and H. Mine, "N-Fail-Safe Logical Systems," *IEEE Trans. on Computers*, Vol. C-20, pp. 536-542, May 1971.
15. Das, S. and Y. H. Chuang, "Fault Restoration Using N-Fail-Safe Logic," *Proceedings of IEEE*, Vol. 60, pp. 334-335, March 1972.
16. Das, S., *Fault-Tolerant Digital Systems Using Fail-Safe Logic*, D.Sc. dissertation, Dept. of Electrical Engineering, Washington University, St. Louis, August 1973.
17. Betancourt, R., "Derivation of Minimum Test Sets for Unate Logical Circuits," *IEEE Trans. Electronic Computers*, Vol. C-20, pp. 1264-1269, November 1971.
18. Dandapani, R., "Derivation of Minimal Test Sets for Monotonic Circuits," *IEEE Trans. on Computers*, Vol. C-22, No. 7, pp. 657-661, July 1973.
19. Spencer, R. F., Jr., "MOS Complex Gates in Digital Systems Design," *IEEE Computer Group News*, Vol. 2, pp. 47-56, September 1969.
20. Lin, T. K., *Synthesis of Logic Networks with MOS Complex Cells*, Report No. UIUCDCS-R-72-517, Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois, May 1972.
21. Reddy, S. M., "Complete Test Sets for Logic Functions," *Proc. of Tenth Annual Allerton Conference on Circuit and System Theory*, University of Illinois at Champaign-Urbanna, October 4-6, 1972.

A study of fault tolerance techniques for associative processors*

by BEHROOZ PARHAMI and ALGIRDAS AVIŽIENIS

University of California
Los Angeles, California

INTRODUCTION—ASSOCIATIVE PROCESSING

Associative processing techniques have been suggested for numerous application areas and have been proven to be superior to more conventional procedures for a number of specialized applications.[1] Recent advances in computer technology and development of new architectural concepts for associative devices have made the design of larger and more flexible systems possible. Such systems are extremely complex and must be adequately protected against failures. This paper reports on the results of a study[2] which has indicated the techniques that are applicable and difficulties that may be encountered in the design of fault-tolerant associative processors.

In the remainder of this section, we will briefly review the four basic organizations for associative processors; i.e., fully parallel, bit-serial, word-serial, and block-oriented. This discussion is motivated by the fact that each of these organizations requires a different treatment for some fault tolerance considerations, such as the detection of failures. This classification is based on the degree of parallelism in operations or, alternatively, the amount of storage associated with each unit of processing logic. A more detailed discussion of these concepts and a comprehensive set of references can be found in Reference 1.

In fully parallel associative processors, processing logic is associated with each bit of stored data. Most fully parallel systems implement only the exact-match search operation in hardware and use software techniques for arithmetic, logic, and more complex searches. An associative processor has been proposed[3] in which a variety of comparison and arithmetic operations are performed in parallel on each word.

In bit-serial associative processors, processing logic is associated with each word of stored data. All the words can be processed in parallel, each in a bit-serial manner. Bit-serial systems represent a compromise between fully parallel and word-serial systems and can be economically implemented with state-of-the-art technology[4] since they can utilize conventional storage elements.

In word-serial associative processors, a single processing unit operates serially on all the words.[5] This approach essentially represents hardware implementation of a simple program loop which is used for linear search. The elimination of instruction fetching and decoding time and the high data rates that can be achieved by circulating memories contribute to the relative efficiency of this approach as compared to programmed linear search.

In block-oriented associative processors, one block of information is associated with a unit of processing logic. A low-cost implementation of such a system may use a head-per-track magnetic recording memory in which each block is stored on one or more tracks.[6] Block-oriented organization is particularly suitable for applications such as information storage and retrieval where a large storage capacity is required.

FAULT TOLERANCE OF ASSOCIATIVE PROCESSORS

Based on the applications that have been proposed for associative devices, there are at least three reasons for studying the fault-tolerance problems of such devices: (1) In some proposed application areas for associative processors, such as air traffic control,[7] the effect of an undetected fault-induced error may be catastrophic; (2) To be able to perform control functions[8] in a fault-tolerant computer, an associative device must itself be fault tolerant, since, otherwise, it will become part of the system's hard core and will contribute heavily to its unreliability; (3) The extreme complexity of large, general-purpose associative processors necessitates the incorporation of fault tolerance features into their design.

It is remarkable, therefore, that the problem of fault-tolerance of associative devices has remained virtually untouched. Ewing and Davies[4] give techniques for coping with some hardware malfunctions in a plated-wire implementation of a particular associative processor. Furthermore, they are only concerned with detecting such errors and disabling the corresponding cell. Fault detection is done by performing certain operations periodically. Proudman[9] suggests that a single error correcting code can be used in conjunction with mismatch detectors with a threshold of 2. However, this scheme is not valid if logic or masked write operations have to be performed, since such operations destroy the coding. Lipovski[10] presents an associative processor architecture in

* This research was supported by the National Science Foundation, Grant No. GJ 33007X.

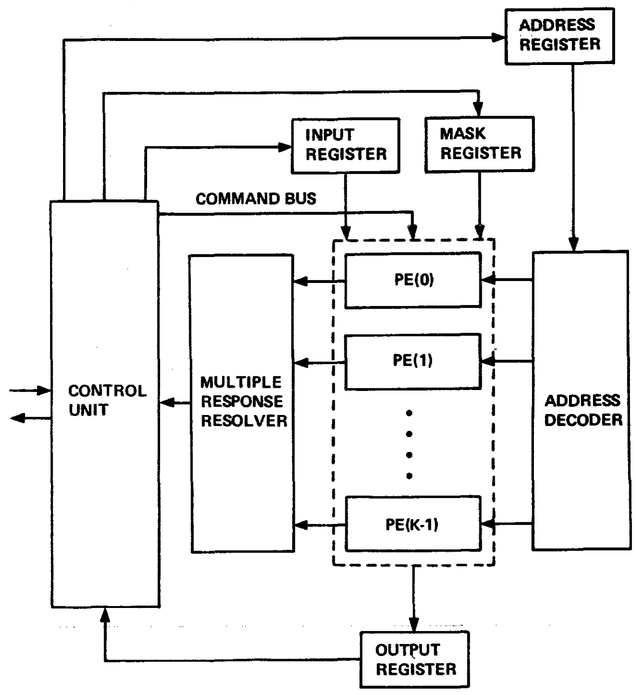


Figure 1—General model for an associative processor

which the processing elements are connected into a tree structure. He contends that such a system is fail-soft since faulty subtrees can be easily isolated from the rest of the system. However, he does not indicate how faults are detected.

In the remainder of this paper, we will identify and discuss some techniques that are applicable in the design of fault-tolerant associative processors. We will concern ourselves with hardware faults and will assume the programs to be correct representations of intended algorithms for the specified domain of operation. We may note, however, that the simplified software of associative processors (e.g., fewer loops), with respect to conventional systems, results in a proportional simplification in the problem of software fault tolerance. A summary of the results presented here has been published elsewhere.[11]

Figure 1 shows a model for an associative processor which applies to all of the classes described in Section 1 except for word-serial systems. Since word-serial associative processors closely resemble conventional systems, their fault tolerance problems can be studied separately. Each processing element (PE) in Figure 1, consists on one unit of processing logic and its associated storage elements. In general, the processing elements in the PE array communicate with each other and the exact pattern of intercommunication is application-dependent.

A study of fault-induced errors in an associative processor shows that they are not easily detectable since a single fault may cause an arbitrary number of errors. This is evident for faults in global subsystems of Figure 1, such as the input and mask registers. For example, in a search operation a smaller, larger, or an entirely different set may respond.[2] A single fault in one processing element may cause errors in others because of PE intercommunication, making concurrent fault detection highly desirable. The problem is further compounded by the fact that each PE performs logic and selective write operations on individual data bits which as we know are not easily checkable without a high level of redundancy.[12]

The selection of applicable redundancy techniques is the most important step in the design of fault-tolerant digital systems. The first basic choice is between static (masking) and dynamic (replacement) schemes. The advantages of dynamic redundancy schemes over static ones are well-known.[13] For associative processors, there are at least two other advantages to the dynamic redundancy approach: (1) The high degree of internal complexity makes the implementation of a statically redundant associative processor very costly and inconvenient; (2) The highly regular structure of a major part of an associative processor (PE array) lends itself naturally to modularization. Such modules can be made identical in structure and can share spare modules.

Let us assume that the associative processor of Figure 1 is divided into M modules, each consisting of P processing elements. Figure 2 shows a possible structure for each module if the decoding and multiple response resolution functions are distributed among the modules. As shown in Figure 2, the information regarding the responses is passed serially through the modules. Clearly fully parallel and mixed series-parallel schemes can be used in much the same way as carry-

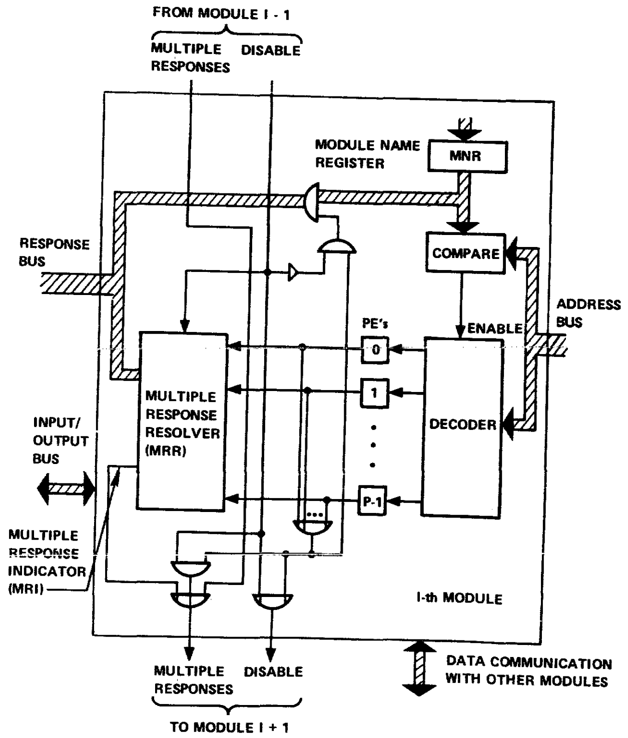


Figure 2—Organization of a module in a modular associative processor

lookahead circuits for adders. Figure 3 shows the modules and their interconnections. One-dimensional intercommunication between modules will be assumed for simplicity.

Given a modular associative device as shown in Figures 2 and 3, it can be made fault tolerant by the following steps: (1) Incorporating internal failure detection ability within each module; (2) Adding S spare modules; and (3) Designing switching mechanisms and corresponding algorithms for reconfiguration. We will assume that the M+S operating and spare modules are permanently connected to the main data buses and that special isolating circuits exist between each module and the data buses. Therefore, reconfiguration takes place by "power switching" and by providing alternate intercommunication paths between modules.

ERROR DETECTION TECHNIQUES

As noted earlier, the problem of error detection in associative processors is a difficult one and conventional coding techniques are generally not applicable. However, there are special cases where low-redundancy coding techniques can be used. We now discuss some such special cases with respect to the four classes of associative processors mentioned earlier. This discussion will be followed by a brief introduction to the self-checking design technique which is applicable in all cases.

A fully parallel associative memory with only "exact-match" search operation and without masking capability can be protected by using a code with a minimum distance of d. With this scheme, if conventional mismatch detectors are used, stored words containing $d-1$ or fewer errors will never respond to a search operation (there is always at least one mismatch signal) and are effectively isolated from the rest of the system until periodic diagnosis routines detect their failure. On the other hand, if mismatch detectors with a threshold of $d \div 2$ are used, up to $k=\lceil d \div 2 \rceil - 1$ bit errors can be masked by the search logic; i.e., a word containing k or fewer errors will still match its original value (k or fewer mismatch signals) and will not match any other value ($k+1$ or more mismatch signals). The difficulty is that such an

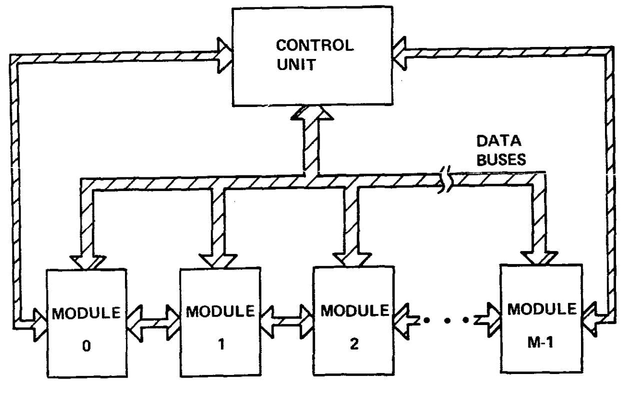
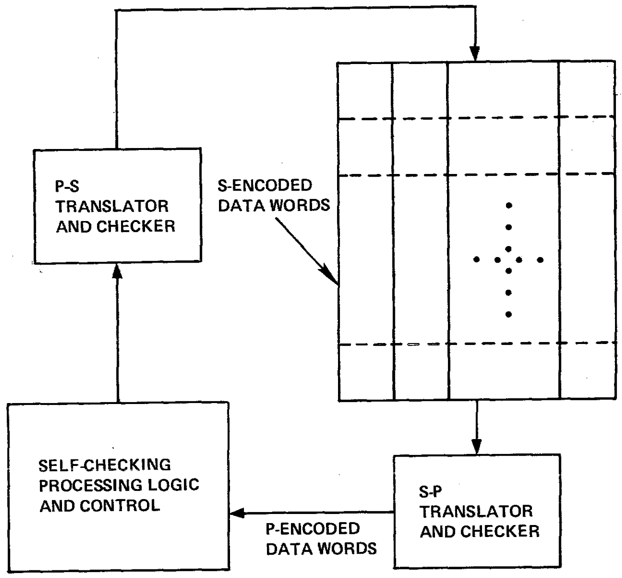


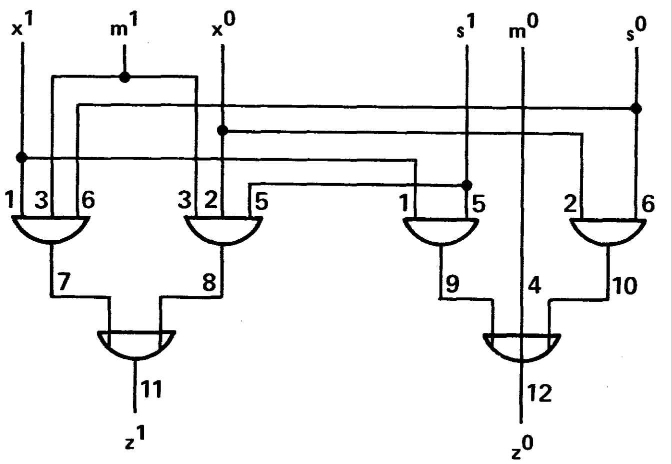
Figure 3—Module intercommunication in a modular associative processor


P-S TRANSLATOR AND CHECKER
S-ENCODED DATA WORDS
SELF-CHECKING PROCESSING LOGIC AND CONTROL
P-ENCODED DATA WORDS
S-P TRANSLATOR AND CHECKER
Figure 4—A fault-tolerant word-serial associative processor

associative device will have no application besides simple table look-up. For most other applications, masking capability and more complex search types are essential. Also, in associative processors, arithmetic and logic operations need to be performed. Clearly, low-redundancy codes are not applicable for such operations.

Considerations for bit-serial systems are similar to those for fully parallel systems. One advantage which exists here is the serial processing of bits in each word. This allows us to artificially extend each operation to the entire word by performing "null" operation on bit positions not originally specified. Now, since all the bits of each word are processed serially, codes with low-cost serial encoding and decoding can be used to protect against storage errors. Simple parity checking is particularly attractive because of the small amount of additional circuitry required for encoding and checking. It should be noted, however, that if operations on small fields within the words are to be performed frequently, the above scheme may result in a significant reduction in speed. Also, operations on multiple fields within the same word (e.g., adding two fields and storing the sum in a third field) do not lend themselves to this approach unless a complete circulation is used for each bit operation, resulting in an almost intolerable speed reduction.

As noted earlier, because processing is performed serially in a word-serial system, protection against failures becomes relatively simple. Low-redundancy coding can be used to protect against storage errors. Failures in the processing logic may be detected through self-checking[14] design. Self-checking translators may be needed to convert the storage encoding (S-encoding) to an encoding suitable for processing (P-encoding). The main requirement on the P and S encodings is that fast (parallel) translation between the two must be possible. This is true since the data rates achieved by circulating memory devices are very high (bit rates of



```
INPUT           TESTABILITY OF LINES
PATTERNS        S-A-1, S-A-0, OR NONE

            |                   1 1 1
1 2 3 4 5 6 | 1 2 3 4 5 6 7 8 9 0 1 2
------------+------------------------
0 1 0 1 0 1 |             1 1     1 0
0 1 0 1 1 0 |     1 0     1 1     1 0
0 1 1 0 0 1 | 1 0     1 0 1 1   0 1 0
0 1 1 0 1 0 | 1 0 0 1 0 1   0 1 1 0 1
1 0 0 1 0 1 |     1 0     1 1     1 0
1 0 0 1 1 0 |             1 1     1 0
1 0 1 0 0 1 | 0 1 0 1 1 0 0   1 1 0 1
1 0 1 0 1 0 | 0 1     0 1 1 1 0   1 0
```

Figure 5—A two-level realization of two-rail masked comparison and its testability with code-space inputs

10-100 MHz). Figure 4 shows a possible configuration for a fault-tolerant, word-serial associative processor. Since during each operation cycle, the entire memory content is circulated through the processing logic, 2-dimensional codes may be used for additional protection against storage errors, if desired.

One favorable property of block-oriented systems with respect to fault tolerance is that during each operation cycle a processing element operates on the entire block of information assigned to it. This enables the use of block codes which result in relatively low redundancy and have simple serial checking algorithms. The simplest possible scheme is to use a parity bit per block of information which detects all single errors. However, if mechanical storage devices are used, error bursts become very probable due to dust particles, minute scratches, or defects in the oxide coating. It has been noted that low-cost arithmetic error codes are very effective for coping with such burst errors.[15] The checking algorithm for these codes is very simple and requires little additional hardware if an adder is already present in each PE.[6]

As can be seen from the previous discussion, low-redundancy coding techniques are applicable only in special cases. Design of logic circuits in self-checking and self-testing form[14] (i.e., in a way that internal circuit failures manifest themselves on the circuit's output and such that each failure is detected by the circuit's normal inputs) particularly if 1-out-of-2 encoding is used appears to be promising. However, because of the relatively higher complexity of the self-checking design approach as compared to low-redundancy coding techniques, this approach should be used when others fail or for protecting the system's hard core.

A detailed discussion of self-checking design concepts is beyond the scope of this paper. Instead, we present as an example a self-checking circuit for masked comparison of two bits. Denoting the mask bit by m, data bit by x and the stored bit by s, the mismatch result z is defined as

$$z = m \cdot (x \oplus s);$$

i.e., we have a mismatch if the given bit position is not masked ($m=1$) and the data bit x does not equal (match) the stored bit s. Figure 5 shows a two-level, self-checking and self-testing realization with two-rail encoding of the variables; i.e., a variable y is represented by a pair (y^1, y^0) with $y^1 = y$ and $y^0 = \bar{y}$ during error-free operation. It is easy to show that any single-line failure results in the correct output or one of the "illegal" combinations (0,0) or (1,1) on the output. Hence, the circuit is self-checking. The fact that the circuit is self-testing is verified by applying an APL/360[16] program called TESTDETECT[17] to it. The table given in Figure 5 is the output of the TESTDETECT program applied to a description of the given circuit. An entry of 1(0) in this table indicates that the corresponding input pattern detects s-a-1 (s-a-0) failure of the given line by producing on the circuit's output one of the "illegal" combinations (0,0) or (1,1). Figure 5 indicates that each line in the given circuit is tested for both s-a-1 and s-a-0 failures during normal operation.

RECONFIGURATION TECHNIQUES

For a modular associative device to tolerate module failures, the module interconnections should not be rigid as shown in Figure 3. Rather, the modules should be interconnected through specially designed switching circuits which prevent a system failure as a result of the failure of a module. The setting of these switching mechanisms determines the

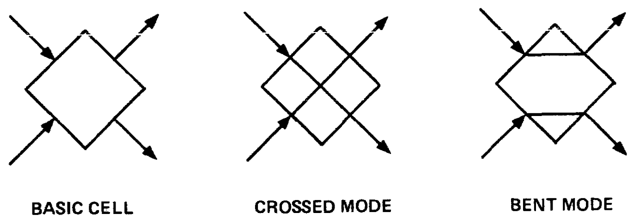


Figure 6—A two-state switching cell

system configuration and can be changed by a central monitor if required. If a module error is indicated and the existence of a permanent failure is determined, reconfiguration procedures must be initiated to establish a new working configuration. In general, data transfers between modules and correction of fault-induced errors is needed as part of the reconfiguration process.

As will be seen the additional complexity introduced by the modularization overhead and reconfiguration switching mechanism is an increasing function of the total number of modules M+S. Therefore, improving the reliability by increasing M and S is possible only to a certain point. Therefore, the optimal module size, in terms of the number of processing elements it contains, and the number of spare modules must be determined for each application through tradeoff studies involving reliability improvement and the corresponding increase in cost.

In the remainder of this section, we will assume only unidirectional (left to right) data flow between the modules in Figure 3. The generalization of the results to bidirectional data exchange is straightforward. After detecting the existence of a faulty module, the following steps must be taken before normal operation can resume: (1) Locating the faulty module; (2) Determining a new working configuration; (3) Initiating appropriate data transfers; and (4) Effecting reconfiguration through switching. The criteria that should be used in evaluating each scheme include: (1) The amount of data transfers needed; (2) The complexity of the reconfiguration algorithms; (3) The number of spares S needed for tolerating f module failures; and (4) The complexity of additional switching circuitry.

A straightforward solution is the use of a "permutation network"[18] which can interconnect the modules in any order. Such a permutation network can be implemented as a cellular array[19] of two-state basic modules shown in Figure 6. Since the complexity of such a cellular permutation network is roughly proportional to the square of the number of modules, its use can be justified only if a relatively small number of modules are involved.

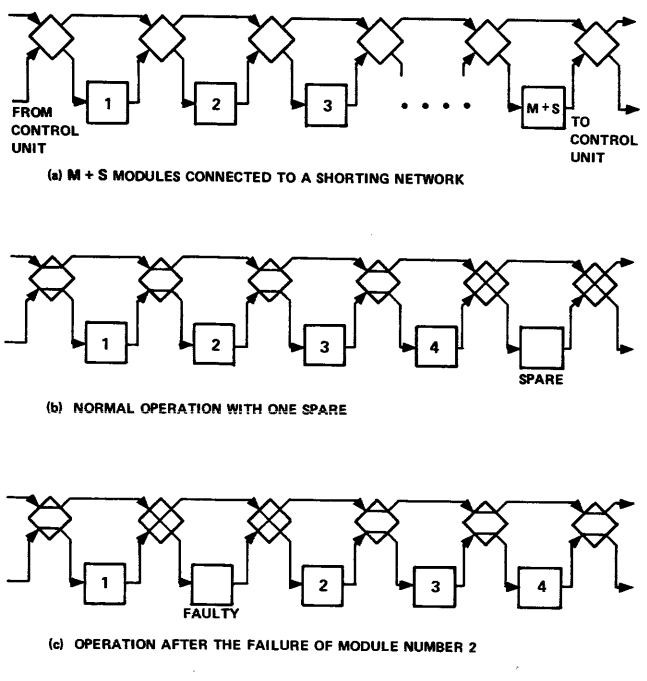

FROM CONTROL UNIT
1
2
3
M+S
TO CONTROL UNIT
(a) M + S MODULES CONNECTED TO A SHORTING NETWORK
1
2
3
4
SPARE
(b) NORMAL OPERATION WITH ONE SPARE
1
FAULTY
2
3
4
(c) OPERATION AFTER THE FAILURE OF MODULE NUMBER 2

Figure 7—Reconfiguration with a shorting network

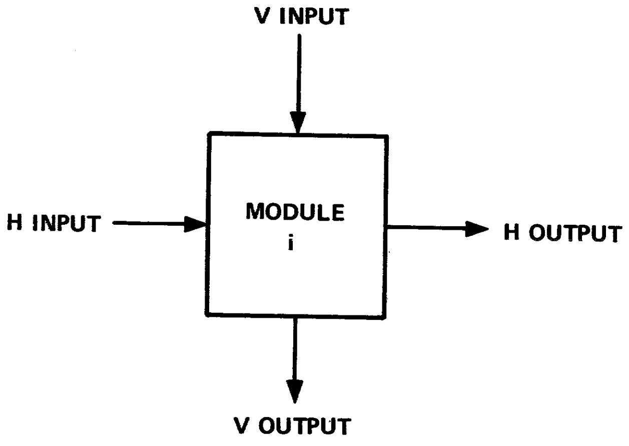


Figure 8—Basic module for distributed reconfiguration

The basic module of Figure 6 can be used in a different way to form a "shorting network."[18] As shown in Figure 7, such a shorting network can be used to route data around the faulty and spare modules. One disadvantage of these schemes, particularly as shown in Figure 7, is the excessive amount of data transfers needed in the case of a failure. The number of transfers needed can be reduced by optimal placement of the spare modules. It can be shown[2] that data transfers are minimized if the kth spare module is in position

$$i_k = k + (k - 0.5) \times M \div S$$

for $k = 1, 2, \ldots, S$. For example, with $M = 6$ and $S = 3$ and the modules numbered $1, 2, \ldots, 9$, the spares should be in positions 2, 5, and 8. Intuitively, this corresponds to dividing the string of M+S modules into S roughly equal groups and placing a spare in the middle of each group. Reference 2 also contains APL/360 algorithms for and examples of reconfiguration with shorting networks.

Another approach to the reconfiguration problem is the use of a distributed switching mechanism; i.e., distributing the switching function among the modules. This can be done by providing each module with a *set* of input and output lines instead of one as shown in Figure 3. Then, if a successor module connected to one module output fails, a module connected to another output can act as its successor. The simplest case, which will be discussed here, is when each module has two sets of inputs and two sets of outputs. As shown in Figure 8, the two inputs and two outputs are distinguished by the letters H and V (horizontal and vertical). The module has four states denoted by HH, HV, VH, and VV, depending on whether the H or V input is used and whether the output is generated on the H or V output.

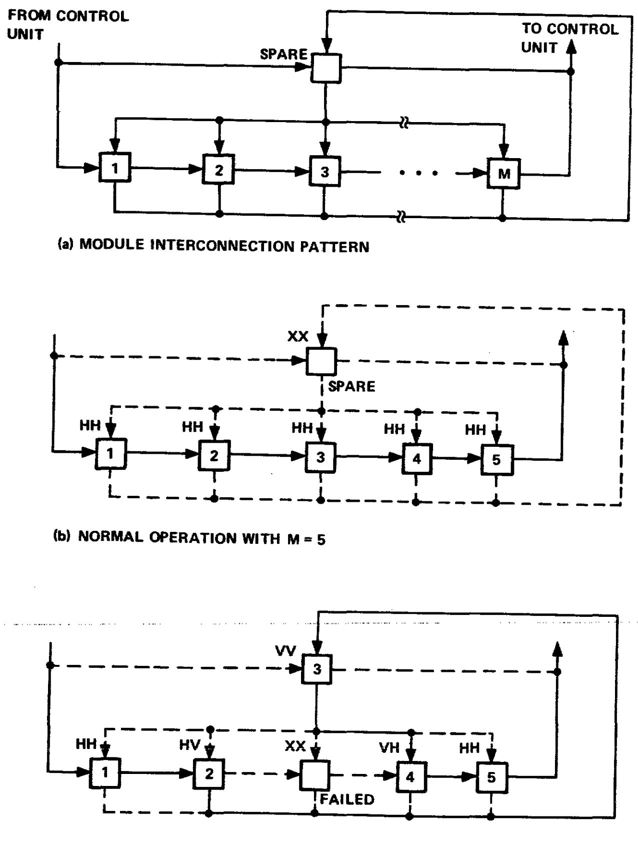

FROM CONTROL UNIT
SPARE
TO CONTROL UNIT
1
2
3
M
(a) MODULE INTERCONNECTION PATTERN
XX
SPARE
HH
(b) NORMAL OPERATION WITH M = 5
VV
HV
XX
VH
HH
FAILED
(c) OPERATION AFTER THE FAILURE OF MODULE NUMBER 3

Figure 9—Distributed reconfiguration with simple sparing

Figure 9 shows how modules of Figure 8 can be interconnected in a simple sparing scheme with $S=1$. The spare module can replace any one of the operating modules and only one module's data need to be transferred in the event of a failure. For $S>1$, this scheme can be used if the M operating modules are divided into S groups each having one spare. The disadvantages of this scheme are: (1) only one module failure can be tolerated in each group; (2) If a faulty module is not reliably powered off, it may produce meaningless data on the common connection to the spare module.

Figure 10 shows a two-dimensional arrangement of the basic modules. It can be seen in Figure 10 that all 9 modules can be connected into a string similar to Figure 3 by appropriate selection of module states. If any single module fails, the remaining 8 can continue their operation. Double module failures will leave at least 6 usable modules. Hence, with $M=8$ and $S=1$, this scheme can tolerate all single module failures. With $M=6$ and $S=3$, all double failures can also be tolerated as well as some triple failures. Note that if both successors of a module fail, it cannot be used. Hence the tolerance of two module failures requires three spare modules. Two interesting and equivalent unsolved problems exist for the two-dimensional arrangement of modules: (1) Given $M+S$ modules with M required to be operating, how should one interconnect them to tolerate the maximum number f of failures? (2) Given the requirement for M operating modules and tolerance of f failures, what is the minimum number S of spares required and the corresponding interconnection pattern?

The basic advantage of this scheme is that the switching mechanism is not part of the system's hard core since a failure in the switching circuits is equivalent to a module failure. The working configuration is supported solely by the non-failed modules. The only place where interference from failed modules may result is on the output bus. This can be avoided by using an output selector circuit to isolate the modules from the bus. The main disadvantages of this scheme are the complexity of the reconfiguration algorithm, excessive data transfers, and tolerance of fewer than S failures. APL/360 algorithms have been written for the reconfiguration process.[2]

It is interesting to note that in a rectangular two-dimensional configuration (Figure 10) with r rows and c columns, one can obtain bounds on the number of modules in various states. Let us denote by n_{HH} the number of modules which

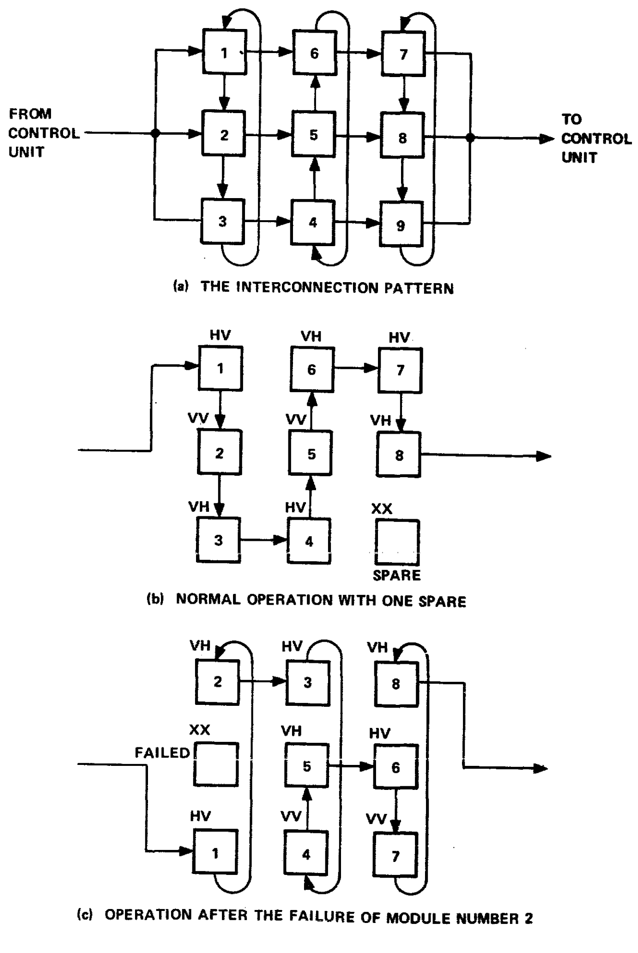

FROM CONTROL UNIT
TO CONTROL UNIT
1
2
3
4
5
6
7
8
9
(a) THE INTERCONNECTION PATTERN
HV
VH
VV
XX
SPARE
(b) NORMAL OPERATION WITH ONE SPARE
FAILED
(c) OPERATION AFTER THE FAILURE OF MODULE NUMBER 2

Figure 10—A two-dimensional arrangement of basic modules

are in state HH. Similarly, define n_{HV}, n_{VH}, and n_{VV}. It can be shown that if M is the number of operating modules, then:[2]

$$\begin{aligned} n_{HH} &\leq (r \times c - M) \div (r-1)(M-c) \div (r-1) \\ &\leq n_{HV} = n_{VH} \leq c\, M - 2c \times \\ &\leq n_{VV} \\ &\leq (M-c) \times (r-2) \div (r-1) \end{aligned}$$

Such bounds are useful in verifying the correctness of a given configuration.

A FAULT-TOLERANT ASSOCIATIVE PROCESSOR

In this section, we illustrate the applicability of some of the techniques discussed previously by presenting the design and evaluation of a fault-tolerant associative processor called SPARE (inverse acronym for Error-tolerant and Reconfigurable Associative Processor with Self-repair). SPARE is essentially a fault-tolerant version of an associative processor which has been described previously.[4] Figure 11 shows a block diagram of the non-redundant system. The random-access memory is used for storing instructions and constants and consists of 4096 24-bit words. The associative memory contains 512 96-bit words. External data can be transferred directly to either one of the memories under automatic interrupt control.

The non-redundant associative processor of Figure 11 can be divided into two parts: (1) The associative (parallel) section, which consists of the associative memory array, bit column selection logic, and word logic; (2) The control and

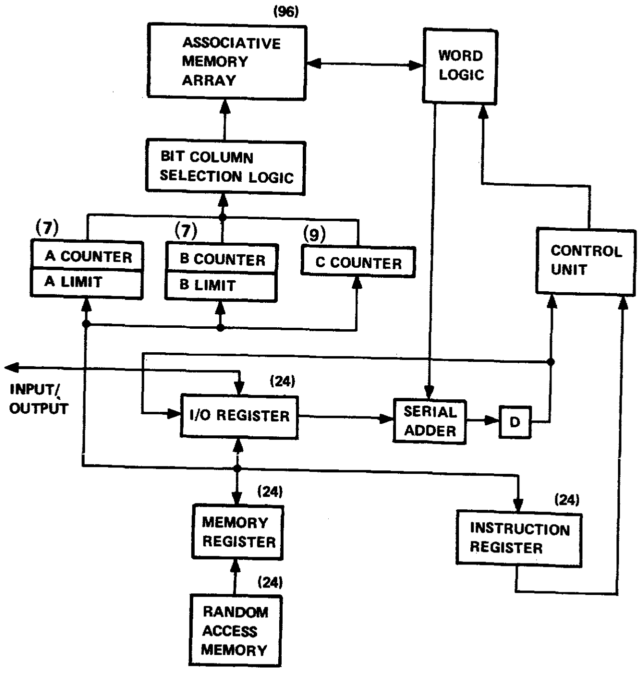


Figure 11—Block diagram of the non-redundant associative processor

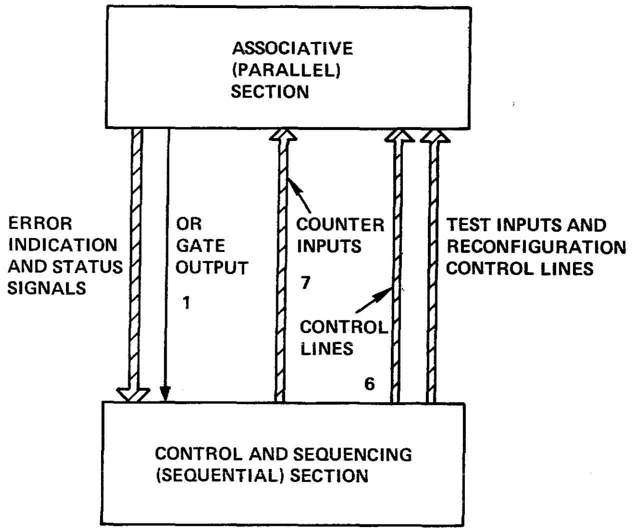

ASSOCIATIVE (PARALLEL) SECTION
ERROR INDICATION AND STATUS SIGNALS
OR GATE OUTPUT
1
COUNTER INPUTS
7
CONTROL LINES
6
TEST INPUTS AND RECONFIGURATION CONTROL LINES
CONTROL AND SEQUENCING (SEQUENTIAL) SECTION

Figure 12—The parallel and sequential sections of SPARE and their interface

sequencing (sequential) section, which contains all other subsystems of Figure 11. Figure 12 shows the parallel and sequential sections of SPARE and their interface requirements. The sequential section uses the status signals and test inputs for monitoring the operation of the parallel section. We now briefly discuss the three main features of SPARE; i.e., error tolerance, reconfigurability, and self-repair.

To achieve *error tolerance*, the parallel section of SPARE is divided into M indentical modules. S spare modules are shared by the operating modules. Each module has internal failure detection capability which is provided by self-checking design of its circuitry using two-rail encoding of logic variables. When a module error is indicated to the sequential section, the recovery mode is entered and the final result may be the replacement of the faulty module by a spare module. The sequential section of SPARE resembles a small general-purpose computer and can, therefore, be made fault tolerant by conventional techniques.

One of the very important properties of associative processors is simple modular growth. The size of an associative processor can grow without a need to alter its algorithms. This suggests that if additional processing capability is required, the redundant processing logic in SPARE can be utilized. Even the two channels of the two-rail circuits can be used independently to double the processing capability if certain design criteria are met.[2] Specifically, we postulate the following operation strategy for SPARE: (1) During normal operation the system works in redundant mode with a number of spare modules; (2) If a module failure occurs or additional processing capability is needed and if a sufficient number of spares are available, they are switched in; (3) If a module failure occurs or additional processing capability is needed and spare modules are not available, the system

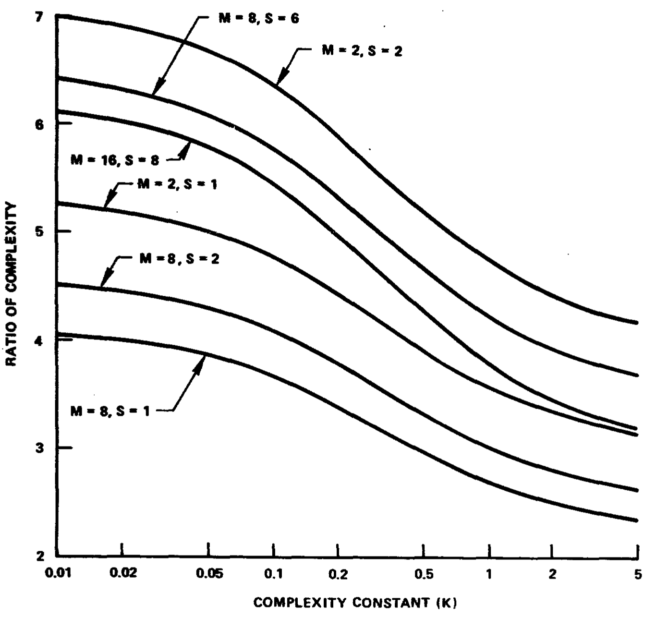


Figure 13—Relative complexity of SPARE as a function of K

reconfigures into simplex mode by utilizing the two channels of the two-rail circuits independently.

Of the reconfiguration techniques discussed earlier, the one using a permutation network seems to be suitable for SPARE since only one intercommunication line (two in self-checking design) exists between modules and the number of modules is expected to be small (M=4 or 8, for example). The *self-repair* process will then essentially consist of computing and setting of a new state for the permutation network. This process must be followed by a recovery procedure to transfer the data stored in the failed module to the one which replaces it. The permutation network has a two-rail self-checking design but no spare is provided for it.

The detailed design of SPARE[2] shows that if K is the relative complexity of one storage bit with respect to a logic gate, the hardware complexity (cost) of various designs, in terms of gate equivalents, are as follows:

Non-redundant system	$\mathrm{NRC}=15872+49152\times\mathrm{K}$
Permutation network	$\mathrm{PNC}=-31+25\times(\mathrm{M}+\mathrm{S})^2$
Each self-checking module	$\mathrm{MC}=512\times(110+192\times\mathrm{K})\div\mathrm{M}$
Redundant system	$\mathrm{RC}=\mathrm{PNC}+(\mathrm{M}+\mathrm{S})\times\mathrm{MC}$

The value of K is technology-dependent and has been chosen as a parameter for generality. Figure 13 shows the ratio of complexity for the redundant and non-redundant designs as a function of K for several configurations of SPARE. In computing the complexity factors, we have only considered the parallel section of SPARE and have ignored the sequential part.

To compute the reliability of SPARE, we will assume that the coverage factor C includes the reliability of the permutation network. Using the reliability modeling technique of Bouricius et al.,[20] we find the following reliability equations directly

$$R_{nr}(T)=\exp(-\lambda_{nr}\times T)$$

$$R_r(T)=\exp(-\mathrm{M}\times\lambda_m\times T)\sum_{i=0}^{S}\binom{\mathrm{M}+i-1}{i}\times\{\mathrm{C}[1-\exp(-\lambda_m\times T)]\}^i$$

where T is the mission time, λ_{nr} and λ_m denote the failure rates for the non-redundant system and a self-checking module, and R_{nr} and R_r denote the non-redundant and redundant reliabilities, respectively. Figure 14 shows the reliability improvement factor defined as $[1-R_{nr}(T)]\div[1-\mathrm{R}_r(T)]$ as a function of mission time T for several configurations of SPARE.

From the preceding discussions we conclude that for mission times which are short compared to the MTBF for the non-redundant system, a significant increase in reliability is possible with a relatively small number of spare modules. A more detailed study of the effect of mission time T, coverage factor C, and complexity constant K on the optimal configuration of SPARE is being carried out. (For a given set of values for T, C, and K, an *optimal configuration* is defined

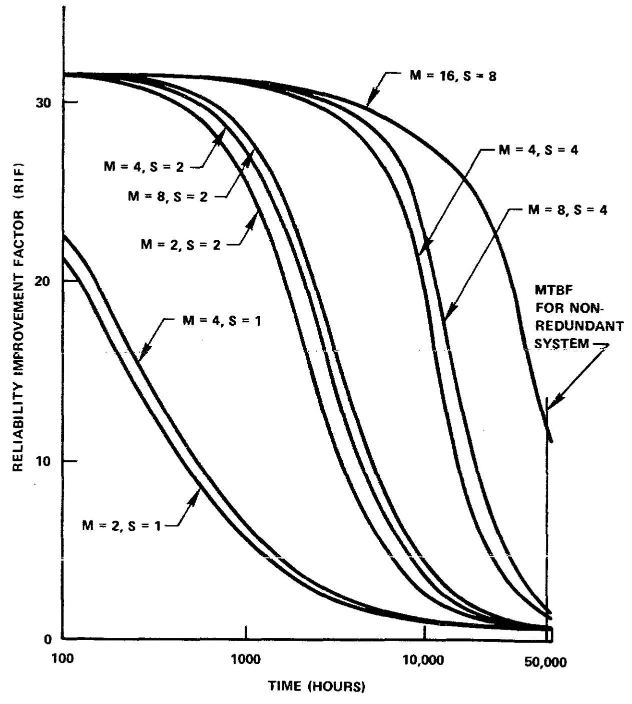


Figure 14—Reliability improvement for various configurations of SPARE with respect to the non-redundant system (K=0.1, C=0.99)

as a pair of values for M and S which result in a lower system cost than all other pairs with the same or higher reliabilities.)

CONCLUSION

In this paper, we have presented the results of a study on the fault-tolerance of associative processors. Our main conclusions are as follows:

(1) Dynamic redundancy is to be preferred over static approach because associative processors lend themselves naturally to modularization and since spares can be shared by a number of identical modules;
(2) Low-redundancy coding techniques are applicable for error detection in associative processors but only in special cases. In particular, the use of arithmetic error codes for block-oriented systems appears to be promising;
(3) Application of self-checking circuit design techniques seems to be an attractive alternative for error detection in associative devices;
(4) Complex switching mechanisms and algorithms need to be devised to enable the sharing of spares by a collection of identical modules which communicate with each other.

Further research is needed in two equally important areas. The first area is the design of completely checked digital circuits. Systematic techniques need to be developed to aid the designers in choosing suitable input and output encodings and producing a self-checking design when presented with a non-redundant circuit or its functional behavior. Cost and effectiveness studies are also needed for the self-checking design approach to determine the increase in complexity over non-redundant designs and the actual error detection coverage which it provides.

The second area is general techniques for reconfiguration in array processors. Extension and generalization of the results presented here are possible in two directions. First, one can conceive of other interconnection schemes for the case where one-dimensional intercommunication exists between modules. For example, we may consider a three-dimensional interconnection pattern in which there are three choices for each of the left and right neighbors for a module. Second, one may seek generalizations to the cases where multi-dimensional module intercommunication is used. This is a considerably more complex problem. As an example, it may be possible to embed a two-dimensional intercommunication pattern into a three-dimensional or four-dimensional matrix of interconnected modules. Then, each module can choose its left, right, upper, and lower neighbors in the same manner as a module could select its left and right neighbors in the case of one-dimensional intercommunication.

Also, we have not considered the testing aspects of associative processors. This is an important area for future investigation since the design of a fault-tolerant associative processor must be initially verified through testing. In addition, for an associative processor which is dedicated to a certain task, there is frequently some idle time which can be utilized by periodic diagnostic routines.

REFERENCES

1. Parhami, B., "Associative Memories and Processors: An Overview and Selected Bibliography," *Proceedings of the IEEE*, Vol. 61, No. 6, pp. 722-730, June 1973.
2. Parhami, B., "Design Techniques for Associative Memories and Processors," Technical Report UCLA-ENG-7321, Computer Science Department, University of California, Los Angeles, March 1972. (Also published as a Ph.D. dissertation.)
3. Shore, J. E. and F. A. Polkinghorn, *A General-Purpose Associative Processor*, Naval Research Lab. Report, Washington, D.C., March 1969.
4. Ewing, R. G. and P. M. Davies, "An Associative Processor," *AFIPS Conference Proceedings*, Vol. 26 (1964 Fall Joint Computer Conference), Spartan Books, Baltimore, Maryland, 1964, pp. 147-158.
5. Crofut, W. A. and M. R. Sottile, "Design Techniques of a Delay Line Content-Addressed Memory," *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 4, pp. 529-534, August 1966.
6. Parhami, B., "A Highly Parallel Computing System for Information Retrieval," *AFIPS Conference Proceedings*, Vol. 41 (1972 Fall Joint Computer Conference), AFIPS Press, Montvale, New Jersey, 1972, pp. 681-690.
7. Thurber, K. J., "An Associative Processor for Air Traffic Control," *AFIPS Conference Proceedings*, Vol. 38 (1971 Spring Joint Computer Conference), AFIPS Press, Montvale, New Jersey, 1971, pp. 49-59.
8. Berg, R. O. and M. D. Johnson, "An Associative Memory for Executive Control Functions in an Advanced Avionics Computer System," *Proceedings of IEEE International Computer Group Conference*, June 1970, pp. 336-342.
9. Proudman, A., "Bulk Associative Memory with Error Correction," *IBM Technical Disclosure Bulletin*, Vol. 12, No. 7, pp. 1076-1077, December 1969.
10. Lipovski, G. J., "The Architecture of a Large Associative Processor," *AFIPS Conference Proceedings*, Vol. 36 (1970 Spring Joint Computer Conference), AFIPS Press, Montvale, New Jersey, 1970, pp. 385-396.
11. Parhami, B. and A. Avižienis, "Design of Fault-Tolerant Associative Processors," *Proceedings of the First Annual Symposium on Computer Architecture*, Gainesville, Florida, December 1973, pp. 141-145.
12. Peterson, W. W., and M. O. Rabin, "On Codes for Checking Logical Operations," *IBM Journal of Research and Development*, Vol. 3, No. 2, pp. 163-168, April 1959.
13. Avižienis, A., "Design of Fault-Tolerant Computers," *AFIPS Conference Proceedings*, Vol. 31, (1967 Fall Joint Computer Conference), Thompson Books, Washington, D. C., 1967, pp. 733-743.
14. Carter, W. C. and P. R. Schneider, "Design of Dynamically Checked Computers," *Information Processing 68*, (Proceedings of IFIP Congress, Edinburgh, Scotland, August 1968), North Holland Publishing Company, Amsterdam, 1969, pp. 878-883.
15. Parhami, B. and A. Avižienis, "Application of Arithmetic Error Codes for Checking of Mass Memories," *Digest of International Symposium on Fault-Tolerant Computing*, Palo Alto, California, June 1973, pp. 47-51.
16. Falkoff, A. D. and K. E. Iverson, *APL/360 User's Manual*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, August 1968.
17. Bouricius, W. G., W. C. Carter, K. A. Duke, J. P. Roth and P. R. Schneider, "Interactive Design of Self-Testing Circuitry," *Proceedings of Purdue Centennial Year Symposium on Information Processing*, Lafayette, Indiana, April 1969, pp. 73-80.

18. Levitt, K. N., M. W. Green, and J. Goldberg, "A Study of Data Commutation Problems in a Self-Repairable Multiprocessor," *AFIPS Conference Proceedings,* Vol. 32 (1968 Spring Joint Computer Conference) Thompson Book Company, Washington, D.C., 1968, pp. 515-527.
19. Kautz, W. H., K. N. Levitt, and A. Waksman, "Cellular Interconnection Arrays," *IEEE Transactions on Computers,* Vol. C-17, No. 5, pp. 443-451, May 1968.
20. Bouricius, W. G., W. C. Carter, and P. R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," *Proceedings of the 24th National Conference of ACM,* San Francisco, California, August 1969, pp. 295-309.

Toward the development of machine–independent systems programming languages*

by KEN MAGEL, ANDRIES VAN DAM and MARTIN MICHEL

Brown University
Providence, Rhode Island

INTRODUCTION

One of the reasons for developing high level languages has been the desire for program portability from one type of machine to another. To achieve the high degree of machine independence necessary for program portability, these languages have included general features such as arithmetic expressions, arrays, and subroutine calls which can be implemented on many machines. Facilities such as the interrupt mechanism, program status word, and device dependent input/output which are available to the assembly language programmer are hidden from the high level programmer. Unfortunately, the code generated by compilers for these high level languages is often very inefficient compared to that produced by experienced assembly language programmers.** However, the added expressiveness and the ability to leave details to the compiler usually offset the inefficiency of generated code, particularly for non-systems applications. For such applications, the facilities which the high level programmer cannot use are not needed anyway.

Systems programming, on the other hand, is one area in which the inefficiency of generated code and the inability to access some features of the machine or operating system make the use of high level languages for entire systems currently extremely difficult. Yet, if a machine and operating system independent systems programming language *could* be developed which permitted the user to access all the facilities of each machine in a natural manner, systems programs could be transported easily from one environment to another. For example, dispatchers, scheduling algorithms, and resource allocation routines could be developed on one machine and then moved to others instead of having to be redeveloped for each environment. Furthermore, systems written in such a language could be studied, and perhaps even debugged without knowing the intricacies and idiosyncrasies of the hardware. Previously developed high level languages especially designed for systems programming such as ECPL[1] and BLISS[2] only partially remove these liabilities. The user still cannot access all the facilities of the hardware in a natural, machine independent manner. Lower level languages such as PL/360 do not solve the problem either because they are too tied to the architecture of one machine. Macro generators such as STAGE2[3] do solve the problem, but require a complete rewrite of each macro when moving from one environment to another. Also optimization across macros which might be possible in some environments cannot be performed.†

This paper describes various approaches which have been taken to developing machine independent systems programming languages. It then explains why none of these provides the full answer. As an example of each approach, certain features of the language for Systems Development,[4,5,6] a PL/I derivative with improved control and data structures, basing modifiers, and string operations, are described. LSD is expressly designed for implementing operating systems, compilers, and other system programs, as well as for large general purpose applications. A prototype compiler for a large subset of the language has been in use for more than two years and generates very efficient code.‡ Presently the full language is being implemented with code generation facilities for the IBM /360-67, /370-168, and two user microprogrammed minicomputers with very different instruction sets (DSC Meta 4's—one programmed as a general purpose machine and the other as a graphics processor). All four machines have specialized features which must be utilized in writing efficient operating systems and in certain other applications.

* The research described in this paper is supported by the National Science Foundation, Grant GJ-28401X, the Office of Naval Research, Contract N00014-67-A-0191-0023, and the Brown University Division of Applied Mathematics.

** Very sophisticated and complex optimizing compilers can sometimes partially improve the quality of generated code; in addition, machines designed specifically for high level languages can somewhat reduce the disparity.

† Very sophisticated and costly macro generators can do some cross-macro optimizations by saving needed information in global variables.

‡ Between 10 percent and 140 percent worse than very carefully hand-coded assembly language depending on the application. These results are from a benchmark study done for the Safeguard Systems Command in 1971.[7] Even the prototype implementation without any sophisticated optimization performed considerably better than either IBM's PL/I-F or Fortran IV H.

PREVIOUS APPROACHES

Traditionally, two general approaches have been taken to the problem of providing access to all of the facilities available in assembly language, but within a high level language: (1) the inclusion of assembly language code for a specific machine within the source program, and (2) the design of high level facilities which are sufficiently general to be meaningful on many machines, but which can be efficiently implemented on each target machine. Although both methods were provided in the first version of LSD, each had serious liabilities.

The ability to include assembly language code within source programs is provided in LSD by two constructs: Code/Endcode for the inclusion of instructions, and Data/Enddata for the inclusion of static data and variables. Items within Data/Enddata are allocated by the compiler in the area generated for static variables of the enclosing block (Begin or Proc) whereas statements within Code/Endcode are placed in the compiler output at a point corresponding to where they appear in the source program. A number of aids are provided by the compiler as illustrated in the example below. For instance, the user may ask the compiler to supply registers in instructions, and he may use names longer than those permitted by the assembler. Qualified names may be used within the assembly code when referring to variables within structures, and array subscripts (constants only) may be used to refer to specific elements of arrays. Code placed within Code/Endcode goes through the full compiler optimization phase* while Data/Enddata is not optimized. Variables used within either of these constructs can be local to the constructs or the same as those referenced in normal LSD statements. An example of the assembler code a programmer could write would be:**

```
CODE:
    L    R?1, T! (2)    place T(2) in a register
    A    R?1, A . B     add B to T(2)
ENDCODE:
```

where T is an array of full words, and B is a full word subelement of a structure named A. A . B is a qualified reference to B, indicating that B is a part of a structure A. The ! is used before the subscript (2) to distinguish the subscript from an index or basing part of the instruction's second operand. The R?1 indicates that the compiler should supply a general purpose register. The number after the question mark is used in determining which uses of question mark registers are meant to be the same register. Given the availability of general register 3, the compiler would treat the above code as if it were

```
    L    R3, T + 4      place T(2) in a register
    A    R3, A + 12     add B to T(2)
```

if twelve were the displacement into the structure A of the substructure B.

* Unless the user specifies that this is not desirable.

** Those unfamiliar with OS/360 assembly language should consult the Appendix.

The inclusion of (even enhanced) assembly language for each machine is tantamount to admitting the impossibility of using high level constructs for all programming. In certain critical portions of the code, the user is forced to program and to think in terms of the low level facilities typical of assembly languages. If the program or the system of which it is a part is to be moved to another machine, the portions of programs which consist of embedded assembly language must be located and rewritten. Often these sections cannot be translated efficiently from one machine's assembly language to another. What is really needed is a rewrite of larger portions of each program.

If there were not some uses which seem to require the Code/Endcode and Data/Enddata constructs, we would eliminate them. However, as will be seen from the discussion of the Convention's statement below, now we do restrict assembly language to macros whose definitions are separate from the program itself.

The other approach is to design general facilities which can be mapped into the special features of the various machines. The design of high level facilities which can incorporate special features of arbitrary machines in a natural manner is extremely difficult, if not impossible. A very sophisticated compiler must be used to try to map general constructs into specific instructions of the target machine and/or specific features of the operating system. For example, IBM System/360 machines have a number of privileged instructions to permit the authorized assembly language programmer to manipulate the program status word, the storage protect keys, and the hardware I/O mechanisms. The first version of LSD included fourteen statements, one for each of these special instructions.[8] These statements could not be implemented for other machines. There seemed no way to provide general high level constructs which captured the power of these instructions in an easily understood manner. In addition, the general constructs should not provide facilities which cannot be provided easily on a System/360. The fourteen statements have been removed from LSD.

THE CURRENT LSD APPROACH

Since neither of the traditional approaches seemed to satisfy our need for an environment independent systems programming language, a different approach seemed necessary. If the code containing environment dependent features could be easily isolated, it could be included within macros. The macros would have to be rewritten for each environment, but if their use were suitably constrained, movement from one environment to another would involve little rewriting and would not affect the appearance and logic of any of the programs. To guarantee that all the environment dependent code was in the macros, the user would have to be able to prevent the compiler from generating any environment dependent code for other constructs. A general macro replacement facility was not developed because it left the decision as to what code would be environment dependent to the individual programmer when he was writing the individual routine. These decisions should be controlled by the project manager or the project design (see the discussion of advantages below).

Accordingly a small number of high level constructs which are present in almost all programming languages were selected. Knowledgeable users can specify alternative implementations for each of the selected constructs. Users can then indicate where each alternative implementation should be used.

The user supplies these alternatives by writing macros which expand into either LSD statements, the assembly language of the machine for which implementation is intended, or a combination of the two. The compiler provides defaults for all of the constructs.*

There are three major areas in the implementation of high level languages where the compiler-generated code must interface with the machine and the operating system. These are input/output, module linkages, and dynamic storage allocation. LSD provides only the most rudimentary input/output.** The high level constructs involved in module linkage are procedure entering, exiting, and returning. Those involved in dynamic storage allocation and release are block (PROC and BEGIN) entry and exit, and the management of based variables (the Allocate and Free statements). Procedure entering, exiting, and calling and the allocation and release of based variables were selected as the subset for which the knowledgeable user will be able to supply alternative implementations. A major advantage of including procedure exit and entry is that anything which could conceivably involve special features of a machine or operating system can be logically programmed as a procedure call. Note that the alternative code which the user supplies can do the function inline, if desired.

THE LSD IMPLEMENTATION

The Conventions statement is used to specify alternative implementations of the linkage and storage management constructs. The syntax is basically that of a macro. The traditional macro facility has two parts: a definition of the macro, and an invocation. Our method involves three parts: (1) the Conventions statement which defines the macro, (2) declarations of the procedures or variables involved which indicate which macros, if any, are to be applied, and (3) the Call or Allocate and Free statements which actually invoke the relevant macro.

The Conventions statement has a labeled header giving the symbolic parameters, followed by a list of parameterized LSD or assembly statements and an End statement. The label on the header is the name of the Conventions statement. As many Conventions statements as desired may be included in a program.

The user specifies where he wants the statement list to be used instead of code normally generated by the compiler† by using the Conventions option on Procedure or Declaration statements.‡ For example, to have the Conventions code generated instead of that normally generated for Procedure entry, a Procedure statement of the following form would be used:§

```
    ADDTC:PROC(LIST, ELEMENT) CONV(TRANS);
```

Here ADDTO is the name of the procedure, LIST and ELEMENT are parameters, and TRANS is the name of a Conventions statement.

To also have different Conventions code generated instead of the normal returning code from this procedure, the following statement would be used:

```
    ADDTC:PROC(LIST, ELEMENT) CONV(TRANS)
      RETURN(CONV(BACK)):
```

TRANS is the name of a Conventions statement which would be used in generating the entrance code for ADDTO. BACK is the name of a Conventions statement which would be used in generating the exit code for ADDTO. In a procedure calling ADDTO a declaration of ADDTO could be used to specify Conventions for calling and after the return from ADDTO:

```
    DCL ADDTO ENTRY CONV(GOIN) RETURN
      (CCNV(CUT));
```

The keyword ENTRY indicates that ADDTO is the name of a procedure or entry point. When ADDTO is called, the statement list in GOIN will be used instead of normal calling code. Immediately following this code will appear the code for the statement list in OUT. As an example consider the following:

```
    RAND: PROC;
          DCI Z ENTRY CCNV(CONE);
          . . .
          CALL Z(B, C, D);
    CONE: CONV P(Q, R, S, T);
          IF Q> R
                THEN R = R + S;
          IF S> T
                THEN T = T + S;
```

* We could have used an extensibility mechanism to provide "new" constructs which would be defined differently in each environment. This is the approach taken by SPECL.[9] Our macro facility is almost identical to Udin's. In our case, however, the decision on when to invoke the macro is not made at the point of invocation.

** All I/O will probably be done in other languages with the LSD routines calling these other routines. I/O is not included in LSD because it is so closely tied to the environment in which a program runs.

† The statement list is substituted during the syntactic phase of the compiler. The substituted code thus goes through all the compiler optimization phases.

‡ Conventions invocations cannot be nested so use of calls or allocates and frees of based variables within the statement list generates the default compiler code in all cases.

§ The syntax of the LSD Procedure statement is almost identical to the corresponding PL/I syntax. The Appendix gives a brief discussion of the portion of LSD used in this paper.

```
        CODE;
                L     R3, Q
                L     R4, R
                L     R5, S
                L     R6, T
                LR    R2, R13
                LA    R13, SAVM
                L     R1, = V(P)
                BALR R14, R1
        ENDCODE;
        END; "END of CONVENTIONS"
```

The call of Z above

```
        CALL Z(B, C, D);
```

would be replaced by

```
        IF B>C
                THEN C = C + D;
        CODE;
                L     R3, B
                L     R4, C
                L     R5, D
                LR    R2, R13
                LA    R13, SAVM
                L     R1, = V(Z)
                BALR R14, R1
        ENDCODE;
```

Notice that any statements involving parameters of the Conventions statement which are not present in the call (T in the example) are not generated. One Conventions statement may be used for any number of different procedure calls by using the name of the Conventions statement in CCNV options on the entry declarations within the calling procedures of the procedures being called.

LSD provides a number of compile-time facilities to aid in writing Conventions statements. The omission of any statements in the statement list which involve parameters not present in the invocation of the Conventions statement (the Call statement) generalizes to the omission of an entire IF-THEN-ELSE, if any part of it uses a parameter which is not present, and to the elimination of entire Do loops if an absent parameter is mentioned in the Do header. Compiler generated temporaries may be explicitly referenced by &Tn where n is a positive integer. The compiler will generate a temporary for each &Tn appearing in the expansion of the Conventions statement. &Tn's using the same n will refer to the same temporary. Special compile-time functions are available for using a variable's length, data type and storage alignment to control what code is generated. For example a compile-time IF, written %IF, can be used to indicate that the code within the THEN clause should be generated only if the length of the variable is a full word and the alignment is at least a half word boundary.

The Conventions construct can also be used for allocating and releasing based variables. For this purpose, any of four options may be specified on the declaration statement for a variable. The ALLOC and FREE options specify Conventions statements which are to be used instead of normal allocating or releasing code when the variable is used in an Allocate or Free statement. The ALLOCIN and FREEIN options can be used on declarations of area variables. These indicate Conventions statements which are to be used whenever a variable is allocated or released within the area. Whenever more than one of these options applies (for example when allocating a variable with an ALLOC option in an area with an ALLOCIN), the ALLOCIN or FREEIN is used. If a list of variables is allocated or released in a single statement, the Conventions code is used once for each variable to which it is applicable. Normal compiler allocating or releasing code is used for other variables in the list. The parameters on Conventions statements used for based variables are the area and its basing parameters followed by the variable and its basing parameters.

Conventions statements for based variable management are used primarily in interfacing with the dynamic storage management routines. The next section deals only with the Conventions statements used with procedures since the use of Conventions statements with based variables is very similar.

THE ADVANTAGES

Using the Conventions statement the programmer is able to completely specify the environment in which his program operates. If he wishes, he may leave some of the environment to the compiler and specify only what is necessary for his program to interface with either the operating system or other portions of the system he is implementing. A system manager can, by placing Conventions statements and declarations in Include files* syntax insure that system conventions are observed. This is one of the advantages of having the decision when and which Conventions to apply at a different place (declarations of variables or procedures) than the actual use.

Conventions statements can be used to supply debugging information and parameter checking which can easily be removed when no longer needed. Coroutines, multitasking, backtracking, and even more general control paths can easily be implemented in an operating system independent manner. Top down implementations in the Dijkstra sense can use Conventions statements to have calls to lower level routines or functions initially generate messages and parameter checks. When the lower level routines become available, the Conventions statements can be removed from the declarations of the routines (in which case the Call statement actually does a call) or changed to macros which perform the functions inline. Functions can be selected logically and a decision to implement them as subroutines or as inline code can be delayed until later. The statement lists within the Conventions statements can be tailored to idiosyncrasies of the machine or operating system.

Using these facilities, some degree of portability can be achieved. Those functions which on a particular machine or

* Include files are external program-segments which the compiler will insert (logically) at requested points in a program before syntax analysis.

under a particular operating system use special facilities do so only through Conventions statements. Experienced programmers can design, code, and debug Conventions statements for each such function. The programs being written for the system then use Calls to "routines" to perform the functions (thereby invoking the associated Convention statements instead of the normal calling code). The Conventions statement either links to (or calls) a routine to perform the function or does the function inline.

SOME CONCLUSIONS

The Conventions facility in LSD isolates the environment dependent code. When movement to another environment is considered, the Conventions statements and the declarations indicating where they are used can be examined to determine if the movement is possible and how difficult it will be. The user, if he wishes, can, by specifying Conventions, prevent the compiler from generating any environment dependent code of its own.

We have not been able to eliminate assembly language altogether, but we have moved it out of the programs and into macros within Include files. The logic and flow of a program can be studied and understood without recourse to any environment dependent code.

It is still something of an art to determine which features should be implemented by Conventions statements. The main advantage of the LSD approach is that decisions on what these features should be need not be made until system implementation time and can be made differently for each system. A user wanting to use a program or system implemented for one machine or operating system in a different environment can examine the Conventions associated with the system to determine if transport to his environment is feasible.

The multi-machine implementation of LSD should be available sometime in 1974. We shall then test the practicality of system portability, possibly by trying to move an interactive graphics system implemented under CP-67/CMS on the /360-67 to one of the minicomputers.

APPENDIX—OS/360 ASSEMBLY LANGUAGE

The general form of a statement is

⟨opt label⟩⟨opcode⟩⟨first operand⟩,⟨second operand⟩
⟨opt comment⟩

Opt indicates that the field is optional. The label, if one is present, must begin in column one. One or more blanks must separate the label, opcode, operands, and comment from each other, and no blanks can appear within the operands except in character string literals. Each operand has the general form:

⟨name⟩(⟨opt base register⟩,⟨opt index register⟩)

If the base and index registers are not present, the parentheses and the comma do not appear. When an operand is a register, neither the base nor the index can be specified.

The instructions used in the examples are:

A ⟨first operand⟩,⟨second operand⟩

The first operand must be a register and the second a main storage location. The second operand's contents are added to the first operand.

L ⟨register⟩,⟨second operand⟩

The second operand must be a storage location. Its contents are placed in the register.

LR ⟨register⟩,⟨register⟩

The second operand's contents are placed in the first operand.

LA ⟨register⟩,⟨second operand⟩

The address of the second operand is placed in the first operand.

BALR ⟨register⟩,⟨register⟩

The first register is loaded with the address of the next instruction following the BALR, and control is transferred to the address in the second register.

LSD

The procedure statement has the following general form:

⟨proc name⟩: PROC ⟨opt parameter list⟩
⟨opt modifiers⟩:

where the BNF for the portions used in this paper is:

⟨parameter list⟩ ::= (⟨parm_list⟩)
⟨parm_list⟩ ::= variable | variable,⟨parm_list⟩
⟨convention modifier⟩ ::= CONV(⟨name of Conventions⟩)
⟨return modifier⟩ ::= RETURN(CONV(⟨name of Conventions⟩))

The Declare statement has the following form:

DCL ⟨variable name⟩⟨opt attribute_list⟩;

The BNF for the only option used in this paper is:

⟨entry attribute⟩ ::= ENTRY⟨opt parameter list⟩

The call statement has the following form:

CALL ⟨procedure or entry point name⟩⟨opt call list⟩;

The ⟨call list⟩ is the same as a parameter list except that each parameter can be an arbitrary expression.

The Conventions statement has a header of the form:

⟨conventions name⟩: CONV ⟨routine parm⟩
⟨opt argument parms⟩;

where ⟨routine parm⟩ is a simple variable, and ⟨argument parms⟩ is a parameter list.

For a description of the entire language, see Reference 3.

REFERENCES

1. Richards, M., "BCPL: A Tool for Compiler Writing and System Programming," *Spring Joint Computer Conference 34*, May, 1969.
2. Wulf, W. A., D. B. Russell and A. N. Habernann, "BLISS: A Language for Systems Programming," *CACM 13*, December, 1971.
3. Newey, M. C., P. C. Poole and W. M. Waite, "Abstract Machine Modelling to Produce Portable Software—A Review and Evaluation," *Software—Practice and Experience 2* 2 1972.
4. Magel, K. *LSD Constructs*, Version 15, Center for Computer and Information Sciences, Brown University, December, 1973.
5. Ulicky, C. and K. Magel, *Language for Systems Development, Version III*, Center for Computer and Information Science, Brown University, August, 1973.
6. van Dam, A., R. D. Bergeron, J. D. Gannon and J. V. Guttag, "Programming Language Comparison Study," Brown University, September, 1971.
7. Henrikson, J. O., and R. E. Merwin, "Programming Language Efficiency in Real-time Software Systems," *Spring Joint Computer Conference*, 40, 1972.
8. Bergeron, R. D., J. D. Gannon, D. Shecter, F. Tompa and A. van Dam, "Systems Programming Languages," *Advances in Computers 12*, M. Rubinoff and F. Alt (eds.), Academic Press, October, 1972.
9. Udin, D. "SPECL: A System Programming Dialect of ECL," *Sigplan Notices*, September, 1973.
10. Wirth, N., "PL/360, A Programming Language for the 360 Computer," *Journal of the ACM*, January, 1968.

LPL—A generalized list processing language

by BILLY G. CLAYBROOK

Virginia Polytechnic Institute & State University
Blacksburg, Virginia

INTRODUCTION

Many programming applications, e.g., symbolic and algebraic manipulation applications and artificial intelligence applications, require the use of list structures. Several list processing languages or languages that provide for list processing have been developed, e.g. LISP 1.5,[5] LISP 2,[1] SLIP,[7] PL/1,[8] ALGOL 68,[6] etc. With the exception of ALGOL 68 and LISP 2, none of these languages provide the user with the capability to define the cell size and/or the configuration of the list cell at runtime (PL/1 attempts to do this with BASED structures but falls short because only one component in a BASED structure can have its dimension altered at runtime). ALGOL 68 has the disadvantage that it is not yet readily available for most users, and it suffers from list tracing complexities during garbage collection. LISP 2 was an attempt to allow variable cell configurations, but the project was aborted.

The language (LPL) described in this paper is an attempt: (1) to provide a generalized list processing language available to a wide range of users, (2) to provide the user with the capability to define cells with varying characteristics that are natural for his applications, (3) to allow garbage collection to proceed easily without the complexities associated with list tracing inherent in list structures with cells having a variable number of links, and (4) to provide the user with a set of statements, e.g., COPY, REMOVE, etc., that perform operations for which the user normally has to generate the code. Anyone who has tried to use LISP 1.5 or SLIP to program solutions to problems whose data are not homogeneous in size and format knows the importance of the existence of languages with the capabilities of LPL and ALGOL 68. Fenichel[2] describes the importance of allowing multiple cell types in list structures. LPL allows the user to concentrate on ideas and avoid the clumsiness of trying to determine a way to represent a structure in a less flexible language. LPL is designed for ease of use and ease of implementation. It is an attempt to exploit the advent of virtual memory systems.

Cells in LPL can be members of many lists; thus, when a cell is inserted (deleted) into (from) a list, a knowledge of which pointers to modify is required. Insertion, deletion, and list tracing are the most difficult problems to handle in languages allowing multiple cell structures. The next section gives an overview of LPL. We discuss the solutions to these problems and give a complete description of the LPL data organization, data types, and statement forms in the third section of this paper.

OVERVIEW OF LPL

An LPL program consists of a sequence of statements separated by semi-colons. The initial implementation of LPL is in PL/1 as an extension to PL/1. Programs written in LPL are scanned during a preprocessor pass, translated into a PL/1 program, and compiled by the PL/1 compiler. LPL requires the use of dynamic storage allocation so the choice of PL/1 was natural for an initial implementation (this implementation in PL/1 was possible only after some imaginative uses of BASED structures).

The existing LPL implementation allows any PL/1 statement to be included in an LPL program. Current design of a second implementation does not include any existing language as a base language, i.e., LPL will exist as an individual processor with the capability to perform arithmetic operations, conditional transfers, etc., in addition to the list processing capabilities described in this paper.

Each LPL cell has certain fields in common that reduce the problems inherent in copying recursive lists, garbage collection, etc. LPL uses a "super list" to hold all nodes allocated by the user and a reference count system for garbage collection; hence, simplifying list tracing considerably and eliminating the marking phase of garbage collection. The use of the super list for garbage collection is explained fully in the next section of this paper.

LPL allows the user to invoke the garbage collector or to release cells individually. One might wonder why we allow the user to do this since LPL basically uses a reference count approach to freeing inactive cells. Normally, in systems using the reference count method, the storage for a cell is released as soon as its reference count becomes zero. When a cell in LPL is removed from all lists by the REMOVE statement, its reference count becomes zero. However, it is possible that the user may want to insert this cell into another list that it was not a member of when the REMOVE statement was executed. In order to allow this flexibility and to allow the user to develop his own garbage collection scheme, if he desires, we simply consider the cell to be inactive until its

Word 1		7
Word 2	RL	# REAL Values
Word 3	IT	# INTEGER Values
Word 4	AL	# ALPHA Values
Word 5	ID	# IDENT Fields
Word 6	RF	# RFCNT Fields
Word 7	LK	# POINTER Fields
Word 8	KY	# KEY Fields

Figure 1—Template structure

storage is released by execution of the RELEASE or COLLECT statements.

DESCRIPTION OF LPL

LPL allows any number of cell types to be defined by the user. Each cell type has a template associated with it to describe its characteristics. LPL allows considerable flexibility in creating list structures since it: (1) allows the designer to define any cell type he desires, (2) allows an arbitrary number of pointers per cell, (3) allows multiple types of values per cell, and (4) provides the user the capability to implement cells with identifier (as in SLIP[7]) and reference count fields (the user identifier and reference count fields are not to be confused with the SYSID and SYSRC fields described below).

Data organization in LPL

Each cell allocated in LPL has the following fields in common:

(1) a type field (TYPE) that contains a pointer to the template describing this type of cell,
(2) a copy bit (COPY) for use in copying lists, especially, recursive lists,
(3) a system reference count field (SYSRC),
(4) a system identifier field (SYSID), and
(5) two link fields, MLP and MRP, that link all cells allocated by the user into a doubly-linked "super list" for use by the garbage collector.

The user cannot access directly any of the five fields listed above.

A description of each of the common fields in a cell is necessary to establish the basic concepts of LPL data organization. The TYPE field points to the template. The template gives the number of components and their type in a cell. Fig. 1 gives an example of the format of templates (the format is similar to that of Fenichel[2]). Besides the common entries ((1)-(5) above) in each cell, the user can select up to seven other types of entries (see next section). Each template consists of a variable number of words (at least two) with the contents of each word, except the first, giving the type and the quantity of that type of entry. Word 1 always contains the number of components in a cell (the number of arguments in the DEFINE statement minus one for the type is the contents of word 1); therefore, the contents of word 1 plus one gives the number of words allocated for the template. Codes in the first half of each word except the first indicate the type of entry (the codes are given in Figure 1). Entries in a template can appear in any order (they actually appear in the order they appear in the DEFINE statement). Templates are not duplicated in LPL.

The COPY bit is used to mark records during the copy process. One problem associated with list processors is garbage collection. We have tried to minimize this problem in LPL by using the system reference count (SYSRC) and the super list. The number of lists in which a cell is a member is maintained by the LPL system in the SYSRC field of the cell. As long as the SYSRC is greater than zero, the corresponding cell is active and is a member of one or more lists (not counting the super list). During garbage collection the super list is traversed and the storage occupied by inactive cells is released. The super list is maintained: (1) to alleviate the system from maintaining a record of pointers to all accessible lists created by the user, (2) to avoid the process of tracing all accessible lists during garbage collection and avoiding the marking phase, and (3) to simplify the implementation of certain functions, such as those performed by the REMOVE and REPLACE statements. List tracing[2] in list structures having cells with an arbitrary number of link fields per cell is a problem to be avoided if possible.

LPL allows the user: (1) to release the storage occupied by cells using the RELEASE statement (provided the SYSRC of the cell is zero), or (2) to invoke the garbage collector by executing the COLLECT statement. The garbage collector is also automatically invoked by the LPL system when about 95 percent of the working space available for allocating cells is depleted. Note that garbage collection in LPL does not require any additional working space and is a very simple process when compared to other garbage collection schemes.[3]

Finally, the SYSID field is used to specify the pointer structure in a cell. This field requires only three bits to specify whether the pointer structure in the cell is singly-linked (SYSID=0), doubly-linked (SYSID=1), left-right-linked (SYSID=2), multi-linked (SYSID=3), or key-linked (SYSID=4). Knowledge of the pointer structure by the INSERT and DELETE statements is necessary for each cell because LPL allows any mixture of the five pointer structures to exist in a single list. Thus, the INSERT and DELETE statements need only to access the SYSID field in each cell, instead of interrogating a template, to determine the pointer structure of the cell. One of the main distinctions between other list processors and LPL is that its INSERT and DELETE statements are provided with the capability to handle any of the four distinct pointer structures automatically.

One assumption is made with respect to the pointer structure in LPL cells. For cells with doubly-linked or left-right-linked structures, LPL assumes the *first pointer* is the *backward* or *left pointer*, and the *second pointer* is the *forward* or *right pointer*. If the user feels restricted with these simple assumptions inherent in the INSERT and DELETE statements, then he has the capability to develop his own procedures to replace them.

It is more difficult to provide generalized insert and delete operations for lists with cells having multi-linked pointer structures. The problem presented by multi-linked cells occurs because there is no automatic way to determine which pointers to modify during insertion and deletion unless specific facilities are provided by the list processor language. The easiest solution to this problem is to let the user develop his own insert and delete procedures and define any constraints that he desires. Another solution used by implementers of multilist file organizations,[4] is to associate a key with each pointer. Therefore, when tracing a list with multi-linked cells, the user can specify, by presenting a key, which pointers to access. LPL provides the user with a choice of either solution. With cells having the key-linked structure, the INSERT and DELETE statements assume the first key is associated with the first pointer, etc. The KEY attribute (discussed in next section) is used for identifying keys in a cell.

Data types and attributes

The language includes the data types REAL, INTEGER, ALPHA (equivalent to PL/1 CHARACTER), POINTER, LIST, STACK, and QUEUE. LPL has attributes NODETYPE, IDENT, RFCNT, and KEY. The declaration

```
DCL   ITYP   INTEGER    NODETYPE,
      ID     INTEGER    IDENT,
      RC     INTEGER    RFCNT,
      KY     ALPHA(5)   KEY;
```

declares ITYP to be an INTEGER variable indicating the type of cell, ID to be an INTEGER variable having the identification attribute (i.e., ID represents an ID field like that in a SLIP[7] cell), RC to be an INTEGER variable having the reference count attribute, and KY to be a five character key. The meaning of each of these attributes will be described further in the next section in the discussion of the DEFINE statement.

Definition of cell structure

The DEFINE statement provides the facility for the user to describe the structure of the cells that he desires for an application. Each time the DEFINE statement is executed it sets up a template (unless one already exists) describing that cell type. Suppose we have the declaration statement:

```
DCL   ITYP   INTEGER NODETYPE,
      RV     REAL,
      IV     INTEGER,
      CV     ALPHA(20),
      LK     POINTER,
      ID     INTEGER IDENT,
      RC     INTEGER RFCNT,
      KY     ALPHA(5) KEY;
```

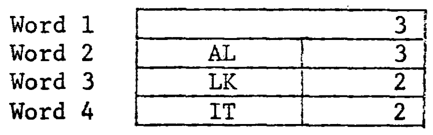


Figure 2—Template structure for cell with three types of fields

The mode of the variables in this declaration statement indicate to the DEFINE statement the types of fields to be included in a cell. Multiple entries of each type can be present in the DEFINE statement (this will be reflected in the template); thus, allowing freedom in *naming* fields in a cell structure. The format of the DEFINE statement is illustrated in the example:

DEFINE CELL (ITYP, RV(H), IV(I), CV(J), ID(K), RC(L), KY(M), LK(N));

The variables in the argument list must appear in a declaration statement preceding the DEFINE statement. The H, I, J, K, L, M, and N INTEGER quantities (they could also be INTEGER constants) indicate how many entries of each type are in a cell. Each cell can have four types of values: REAL, INTEGER, ALPHA, and POINTER. The IDENT and RFCNT entries in a cell allow the user to have his own identifier and reference count fields. These two types of entries increase the flexibility of LPL by allowing the user to develop any list structure format that he desires. We expect that only one identifier and one reference count field will be included in a cell, but the facility exists for more than one of each. Not all variables must appear in the DEFINE statement. The statement

DEFINE CELL (ITYP, CV(3), LK(2), IV(2))

describes a cell of type ITYP with three alphanumeric values (with 20 characters each), two pointer fields, and two integer values. Figure 2 shows the template corresponding to the above cell structure. When a variable with the KEY attribute is included in the DEFINE statement, this implies the cell has a key-linked pointer structure. The number of keys and pointers must be the same. A KEY variable can be any of REAL, INTEGER, or ALPHA data types. The arguments in the DEFINE statement can appear in any order.

LPL data manipulation statements.

We provide LPL with enough high level manipulation statements to save the user time in writing programs, but we also give him the full capability of a list processor to program at a lower level. The tokens in each LPL statement are separated by commas with a blank following each keyword. LPL has the following data manipulation statements (a cell pointed to by POINTER variable P is designated as cell P; and [ ] indicates the contents are optional):

1. GET P,I; — Allocates a cell P of type I.
2. DELETE P,T[(KEY)] [,R]; — Deletes cell P from list T, cell P follows cell R (optional); if key KEY (optional) is

specified then only pointers associated with KEY are modified.

3. INSERT P,T[(KEY)],R; — Same as DELETE except cell P is inserted in list T after cell R (KEY is again optional).
4. RELEASE P; — Frees the storage occupied by cell P. The SYSRC for cell P must be zero.
5. ENTER exp,T; — Enters the value of exp into T (T must be a STACK or QUEUE and the mode of the value of exp must agree with that indicated in the CREATE statement for T).
6. TAKE X,T; — Removes an entry from T (T must be a STACK or QUEUE) and place it in X.
7. RV(N),P = exp; — Sets the *N*th REAL, INTEGER, ALPHA, POINTER, IDENT, RFCNT, or KEY field in cell P to the value of exp.
8. X = RV(N),P; — X becomes the *N*th value of one of the fields in cell P.
9. COPY S,T; — Copies list S into list T.
10. P = N,T[(KEY)]; — P is the pointer (optionally associated with key KEY) to the *N*th cell in list T.
11. SPLIT T [(KEY)],P; — Split list T at cell P. The use of KEY is optional like in the DELETE statement.
12. CONCAT S[(KEY)], T; — Concatenates lists S and T in the order given.
13. COLLECT; — Invokes the garbage collector.
14. ERASE T; — Erases list T (cells are deleted from T).

```
15. P, (SINGLY )
       (DOUBLY )
      {(L-R    )};— Declares cell P to have one of the
       (MULTI  )   five pointer structures indicated.
       (KEY    )
16. CREATE T (STACK){SINGLY}  (REAL   )
                    {DOUBLY}  (INTEGER);—
            {QUEUE }{SINGLY}  (ALPHA  )
                    {DOUBLY}  (POINTER)
             (LIST ){SINGLY}
                    {DOUBLY}
                    {L-R   }
                    {MULTI }
                    {KEY   }
```

Declares T to be a pointer to a specific structure. The data types REAL, INTEGER, ALPHA, and POINTER apply only to STACKS and QUEUES.

17. REMOVE P; — Removes cell P from all lists of which it is a member.
18. REPLACE P,R[,T[(KEY)]]; — Removes cell P from all lists or from a single list T and replaces it with cell R. Only the pointers associated with KEY may be changed (optional).

A few of the LPL statements require a more detailed description; however, some such as COPY, CONCAT, etc. are self-explanatory and need no further discussion (algorithms for COPY, CONCAT, etc. are either well-known or trivial and are not discussed). The GET statement allocates a cell of a specific type. During the allocation of this cell, the template for this cell type is interrogated to determine the cell's structure. Statement #15 above is required to fix the pointer structure of cells in a list not created by the CREATE statement. Failure to do this results in a cell having a singly-linked pointer structure automatically assigned to it. The INSERT statement increases the SYSRC by one for each execution, and the DELETE statement automatically decreases the SYSRC by one (other information about these two statements has been discussed in detail in an earlier section). Statement #7 is important because it allows the user to place values in the respective fields of cell P.

The ENTER and TAKE statements operate on STACK and QUEUE structures defined by the CREATE statement. Before discussing these two statements it is necessary to describe the CREATE statement. The CREATE statement allows the user to set up LIST, STACK, or QUEUE structures with a homogeneous pointer structure. The variable T in the CREATE statement is of type POINTER and points to a descriptor. The CREATE statement clearly describes the format of the defined structure. The user does not have to set the pointer structure using statement #15 in each cell for these structures.

At compile time the code is generated to allocate and initialize each descriptor. Descriptor pointer fields are set to NULL. The format of these descriptors is given in Figure 3. The first word in each descriptor indicates the type of structure being described (actually the first word is not an alphanumeric string as indicated but is an integer identifier). The second word describes the pointer structure. The third word in both the STACK and QUEUE descriptors indicates

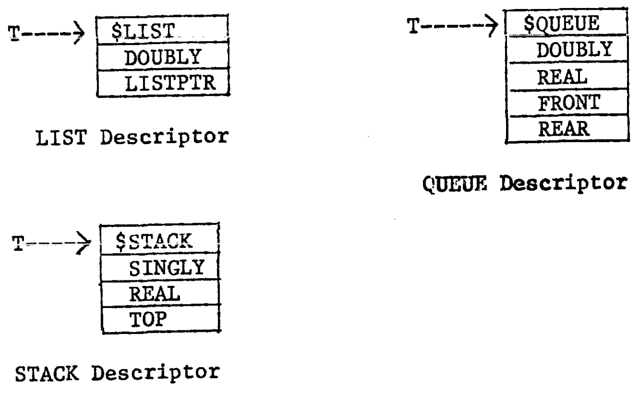


Figure 3—LIST, STACK, and QUEUE descriptors

the mode, e.g., REAL, of the values in the structure. The last word (or words) of each descriptor point to the structure.

At runtime the pointer entries in Figure 3 are updated and modified as the structures are manipulated. As items are entered (removed) into (from) STACKs and QUEUEs, the pointers TOP, FRONT, and REAR are updated. Associated with STACK structures is a function TOP(T) that returns as its value the pointer to the top of stack T. FRONT(T) and REAR(T) are functions whose values are pointers to the front and rear, respectively, of QUEUE T. HEAD(T) is a function associated with LISTs. Its value is the pointer (LISTPTR) to the first cell in the list. These functions can only be used with structures declared in the CREATE statement unless the user sets up his own descriptors with the same format at LPL.

The ENTER and TAKE statements enter (remove) information into (from) the stacks and queues. The pointers to the TOP, FRONT, and REAR of the defined structures are automatically updated by the ENTER and TAKE statements.

The only other statements that we discuss are the REMOVE and REPLACE statements. Both are basically the same with the exception that REPLACE inserts another cell in place of the one removed. These statements perform several operations that would otherwise require a considerable amount of coding on the part of the user. At a given time there is no way to know in which lists cell P is a member (unless some very expensive, with respect to storage, bookkeeping tasks are carried out). But we do now the *number* of lists in which cell P is a member (by using the SYSRC value). There are two ways to attack this problem. The first way is to begin tracing the user created lists and determine in which lists cell P is a member. But as we have stated before, list tracing in lists with cells having a variable number of pointers is difficult and time consuming. Also storage is required to save pointers during the tracing. The second way (done by the REMOVE and REPLACE statements) is to trace the super list and determine which cells contain pointers to P. The examination of cells in the super list ceases when LPL locates the number of cells (containing pointers to cell P) given by the value of the SYSRC field in cell P. In general, we expect less time to be consumed in tracing the super list (because of its simple pointer structure) than other user created lists; and no extra storage is required to save pointers.

SUMMARY

LPL is an attempt at providing programming convenience for the user. LPL is different from most other list processing languages because it allows the user to define multiple cell structures and cell sizes at runtime, thereby allowing list structures with multiple cell-types. LPL allows the user much flexibility in defining list structures. This capability proves useful when the user's data are not homogeneous. It also allows the user to represent structures in a more natural manner. While LPL is a flexible language that can be tailored to the user's requirements, it provides the complete and convenient programming facilities of a ready-made system.

List tracing is the primary obstacle, especially during garbage collection, to implementation of list processing systems allowing multiple cell types. LPL uses a super list containing all cells previously allocated and a reference count scheme to make garbage collection trivial. No marking phase is required; hence, complex list tracing is avoided. One of the primary design features of LPL is to avoid list tracing as much as possible.

LPL statements provide for handling the normal list processing operations, e.g., copying recursive lists, erasing lists, splitting lists, concatenating lists, garbage collection, etc. Other operations such as removing a cell from some or all of the lists of which it is a member (and possibly replacing it with another cell) are performed automatically by the execution of a single statement. All LPL statements can directly handle singly-linked, doubly-linked, left-right-linked, and key-linked pointer structures. Special facilities are provided for the creation of stacks and queues.

Finally, LPL has been used by the author in artificial intelligence problems and in polynomial manipulations. These two problem areas require the use of several cell types.

REFERENCES

1. Abrahams, Paul W., "The LISP 2 Programming Language and System," *AFIPS, 1966, FJCC*, Vol. 29, Spartan Books, N.Y., pp. 661-676.
2. Fenichel, Robert R., "List Tracing in Systems Allowing Multiple Cell-Types," *CACM*, Vol. 14, August, 1971, pp. 522-526.
3. Knuth, D. E., *The Art of Computer Programming*, Vol. 1, Addison-Wesley, Reading, Mass., 1968.
4. Lefkovitz, David, *File Structures for On-Line Systems*, Spartan Books, New York, 1969.
5. McCarthy, John, et al., *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, Mass., 1965.
6. Peck, J. E. L. (ed.), *ALGOL 68 Implementation*, North-Holland Publishing Company, Amsterdam, 1971.
7. Weizenbaum, J., "Symmetric List Processor," *CACM*, Vol. 6, September, 1963, pp. 524-544.
8. IBM System/360 PL/1(F) Language Reference Manual, GC28-8201-4.

Generalized structured programming

by JOHANNES J. MARTIN

Virginia Polytechnic Institute and State University
Blacksburg, Virginia

INTRODUCTION

Structured Programming as defined by Dijkstra[4] produces programs whose flow can be described by one of the basic flow graphs shown in Figure 1.

Each square box in such a flow graph represents either a primitive operation or another well structured program. Thus, Structured Programs are recursively decomposable with respect to these basic flow graphs. As a main characteristic each basic flow graph has only one entry point and one exit.

Böhm and Jacopini[2] who have investigated a very similar set of flow graphs have demonstrated that their set, which is even slightly simpler than Dijkstra's, is sufficient for defining all algorithms. Many Programming Languages include control constructs (e.g., the *if-then-else* or the *while* construct in ALGOL 60 and PL/1) that support, to some extent, the use of the basic flow structures of Structured Programming. Wirth's languages PASCAL,[11] an ALGOL derivative, provides control constructs for all flow graphs needed in Structured Programming (Table I) and no additional ones.

Because PASCAL matches the demands of Structured Programming so completely, the notational extensions suggested in this paper will be stated as extensions of PASCAL.

The main assets of Structured Programs are their clarity and relative simplicity. In particular, the method of Structured Programming encourages top down analysis[9] of problems or the development of algorithms by stepwise refinement.[10] Programs developed in this manner are automatically modular; hence, Structured Programming provides a systematic way of modularizing. Furthermore, as Dijkstra[3,4] has pointed out, structured programs display a simple relation between the progress of the computation and the progress through the program text. As a result, the amount of information needed for determining the computational progress accomplished at some point in a program does not depend on the length of the program (since one does not need a trace) but only on the depth of loop nesting and the depth of subprogram referencing at the point of interruption. All these points clearly enhance the transparency of a program.

Nevertheless, many programmers feel that the method of Structured Programming is too restrictive. In particular, since loops can have only one exit (Figure 1) some simple and very common program structures are outlawed. The classic example is the search loop.[6] The search loop either finds the item wanted, in which case some action A should be taken, or it does not find the item, in which case an alternative action B should be executed. Such a search loop can obviously be interpreted as a program segment that computes a condition and, by virtue of its two exits, selects one of two consequent actions. Thus, such a loop could logically replace a decision box in a flow chart. However, the rules of Structured Programming require that decision nodes are primitive; the substitution of a decision node by a program segment is not permitted since the flow of such a program segment would not conform with any of the four basic flow graphs. Thus, Structured Programming, in effect, classifies conditions which are used to control the flow of a program into two categories:

(a) Simple conditions that can be specified in condition boxes because they can be computed by a single expression, and
(b) Complex conditions that must be computed in a program segment that precedes the test which ultimately makes the selection because they *cannot* be computed by a single expression.

This distinction causes programming steps that are motivated solely by structural restrictions imposed by rules of style rather than by the inherent logic of the problem to be solved. At this point, the generally beneficial rules of Structured Programming definitely lower the understandability of a program.

Shortcomings of Structured Programming have been discussed in the literature, especially in some articles concerned with the elimination of the *goto*-statement.[5,7,13] However, to my knowledge the problem has never been stated as a problem of discrimination among simple and complex conditions, although it seems to be this discrimination that causes the alleged inconveniences of *goto*-less or Structured Programming.

In this paper we shall suggest a set of generalized flow structures as a basis for structuring programs. The main idea is to have basic flow graphs with multiple exits so that not only action nodes (square boxes) but also decision nodes can be replaced by program segments. As a result the artificial distinction between simple and complex conditions will disappear.

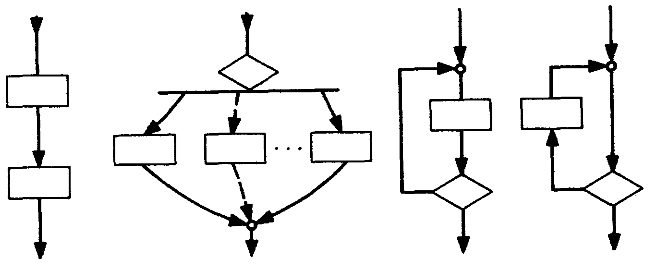


Figure 1—Primitives of structured programming

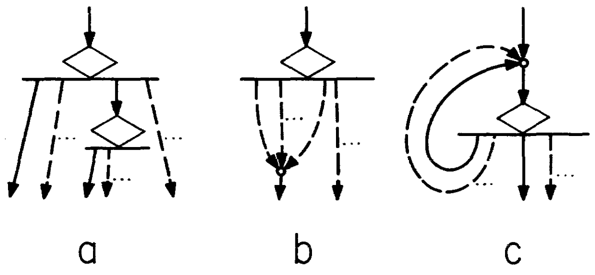


Figure 2—Primitives of generalized structured programming

Allen and Cocke[1] have developed a method for analyzing flow graphs which they call interval decomposition. With this method, one can reduce flow graphs that do not contain multiple entry loops to a single node. We shall be able to show that our set of basic flow structures permits the construction of exactly these flow graphs.

We shall further explain why the advantages claimed for the original system of Structured Programming are maintained by the generalized version suggested. Finally, we shall suggest a set of new control constructs (as an extension of PASCAL) which supports our generalized set of basic flow graphs.

GENERALIZED STRUCTURED PROGRAMMING

We define a well-structured flow graph as a graph that can be described by one of the structures shown in Figure 2 where the nodes represent either primitive operations or well-structured flow graphs. The flow graphs shown in Figure 2 contain only two types of nodes: nodes with several entry points and one exit (later referred to as collector nodes) and nodes with one entry point and several exits (later referred to as action nodes).

Collector nodes do not correspond to any computational action; their function is comparable with that of labels in programming languages. Hence, collector nodes are not further decomposable.

Action nodes have $k \geq 1$ exits and, thus, may occur as decision nodes as well as single-exit action nodes.

Our structures show some resemblance to Dijkstra's sequential (2a), selective (2b), and iterative (2c) modes of operation. However, they are less restrictive since the set of graphs which they generate contains the set generated by Dijkstra's structures as a proper subset. Later we shall refer to our structures as forms 2a-2c; we shall further use the term decomposable (reducible) restrictively for graphs that are decomposable (reducible) with respect to our system.

TABLE I

if⟨expression⟩*then*⟨statement⟩[*else*⟨statement⟩]
case⟨expression⟩of⟨const⟩:⟨statement⟩[;⟨const⟩:⟨statement⟩]* *end*
while⟨expression⟩*do*⟨statement⟩
repeat⟨statement⟩[;⟨statement⟩]* *until*⟨expression⟩
for⟨var.⟩: = ⟨expression⟩⟨sep.⟩⟨expression⟩*do*⟨statement⟩
⟨sep.⟩: : = *to*/*downto*

PROPERTIES OF DECOMPOSABLE FLOW GRAPHS

First we shall show that a graph is decomposable if it does not contain multiple entry loops.

1. We shall consider only flow graphs with exactly one entry point. The node R by which a flow graph is entered will be called its *root.*
2. We postulate that in all flow graphs considered, there is a path R-N for every node N of the graph.
3. Definition: A *loop* (or strongly connected region) is a set S of nodes such that for each pair of $N,M \in S$ there is a directed path N-M.
4. Lemma: A flow graph that does not contain loops can be reduced to an action node by the forms 2a and 2b.
   (Proof by induction based on 1. and 2. using the fact that the nodes in a loop-free graph (3.) are partially ordered.)
5. Definition: Flow graphs are equivalent, if they can be transformed into each other by collapsing adjacent collector nodes or, reversely, by splitting a collector node into two adjacent ones.*
6. Definition: A collector node C is a *selection collector* if there is a node D such that every directed path C-C contains D and there is a simple path D-C for every arc entering C.
7. Definition: A collector node that is not a selection collector is called a *loop head.*
8. Lemma: Every loop contains a loop head. (Follows from 2., 3., 6., 7.)
9. Definition: A loop head L is the head of an *S-loop* (single entry loop) if there exist paths L-L that are

* Since collector nodes do not represent any action, this transformation trivially does not change any essential property of the flow graph (it is equivalent to the introduction of an additional label for an already labeled point in a program). However, it facilitates the creation of flow graphs for which collector nodes can be classified into (a) selection collectors; (b) loop heads for single entry loops; (c) loop heads for multiple entry loops.

disjoined from all paths R-L except for the node L. Nodes *belong* to the S-loop if they are contained in such a path L-L. Other loops are called *M-loops* (multiple entry loops)

10. By splitting the loop head (5.), if necessary, we shall always make sure that all arcs but one entering the head of an S-loop participate in the S-loop. Thus, we shall make sure that only one arc enters an S-loop from the outside.
11. Lemma: If an S-loop L_1 contains the head of an other S-loop L_2, all nodes of L_2 are also nodes of L_1. (Follows from 9.)
    Note 1: Definition 9. assures that the converse of 11. is not true, i.e., there are nodes in L_1 that are not nodes of L_2.
    Note 2: It follows from 11. that an S-loop is not necessarily a proper nest of loops, say, in the FORTRAN sense. Figure 3 gives an example.
12. Definition: The *rump* of an S-loop is the graph that consists of all nodes of the S-loop but the head.
13. Lemma: The *rump* of an S-loop is a graph with one entry point and one or more exits. (Follows from 12. and 9.)
14. Lemma: An S-loop that does not contain any loop, can be reduced to an action node.
    (Proof: The rump is a loop free graph with one entry point (13.) and, hence, reducible because of 4. The loop head can then be attached by form 2c to form a graph with one entry point and one or more exits because of 9. and 10.)
15. Lemma: S-loops that do not contain M-loops can be reduced to an action node.
    (Proof: by induction based on 14., partial ordering is assured by 11.)
16. Theorem: Flow graphs that do not contain M-loops are well structured with respect to the forms 2a-2c.
    (Proof: follows from 4. and 15.)

Because the set of flow graphs that can be constructed with the forms 2a-2c is considerably larger than the set that can be generated from Dijkstra's primitives, one might suspect that some of the advantages of the original set are lost. We believe that this is not the case. The advantages of Structured Programming seem to be based on the decomposability of the programs created and on the fact that textual progress and computational progress are related in a simple way. Both properties are maintained in our system.

(1) By definition, programs are decomposable with respect to our basic control structures. The functions of these structures are clearly not any more difficult to understand than the functions of Dijkstra's primitives: Form 2a and 2b can be understood by enumerative reasoning[4] whereas 2c, i.e. the loop, can be analyzed by inductive reasoning. If formal program verification is persued, we shall need to determine one assertion for every exit of such a basic structure; whereas, for Dijkstra's primitives only one assertion is needed

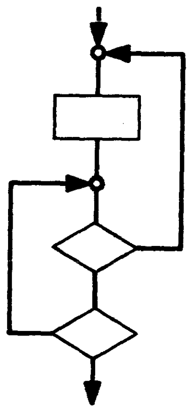


Figure 3—Example of an S-loop

(since there is only one exit). On the other hand, the assertions needed for our structures will frequently be slightly simpler since we shall not need control variables that keep track of computed conditions.

(2) The relation between textual and computational progress is similarly simple in both systems.
    Loop free single entry graphs have a fixed and, thus, trivial relation between computational progress and textual progress.
    Further, all loops are entered only at the loop head for every turn through the loop. Hence, a counter incremented when the loop head is passed, can keep track of a loop in progress. The rump of a loop is again either a loop free single entry graph or it contains other single entry loops. The number of active counters depends, as in Dijkstra's system, only on the depth of loop nestings.

We conclude that our set of primitives leads to similarly transparent programs as do Dijkstra's.

There is, however, one major difference that makes manipulating our flow graphs somewhat more difficult. Structured Programs in Dijkstra's sense are unambiguously decomposable if one ignores the fact that the order in which concatenated actions are associated is arbitrary. This ambiguity does not matter at all because it does not cause any uncertainty since the actions are fully ordered.

For our system, this very ambiguity causes somewhat of a problem: Because we concatenate multiple exit nodes rather than single exit nodes, the structure created by repeated concatenation is only partially ordered. Thus, the way in which the parts are put together makes a structural difference. Nevertheless, if suitable language notations are provided that permit the realization of the primitives as program text, the programmer can (and must) indicate how he wants the operations to be grouped. The programmer's wishes are lost, though, after the program is compiled. However, this fact cannot be counted too heavily against our system since the clarity of source programs and not that of object programs is the main goal.

NEW CONTROL CONSTRUCTS

Before introducing the new control constructs, we should note that there are at least two known constructs that support, to some extent, our system:

(1) Wulf's *leave* feature in his programming language BLISS,[12] and
(2) The *exit* feature as available in ALGOL 68.[8]

These features are particularly effective because in both languages statement compounds may deliver values. Hence, statements may be inserted into conditional statements. The *leave* (*exit*) statement facilitates the termination (completion) of the inserted statement at any point within this statement.

Although both constructs provide quite flexible mechanisms for dealing with complex conditions, none of them fully supports the general loop structure (form 2c); further, the *exit* construct only permits transferring control to the next higher block level. Contrarily, the new constructs suggested fully support our system; also, they do not require that statements possess values.

Two types of control constructs are considered; the first type (A) is a generalization of existing constructs. It has been designed to facilitate the definition of arbitrary conditions within conditional statements. The second construct (B) has been added in order to completely support form 2c.

(A) As an extension of Wirth's programming language PASCAL, we suggest the notion of a group, i.e. a compound statement with potentially multiple exits. These exits are exploited by inserting the group into a conditional statement in the place of the usual expression. Therefore, a group should behave like an expression, i.e. it should possess a value. A group is formed by the brackets

begin—succeeds,
begin—fails, or
begin—group.

The first two forms are used to denote boolean groups, i.e. groups that can be used instead of boolean expressions. The last form is non-boolean and can be used in the *case* construct.

Examples:

```
(1) while begin⟨statement⟩succeeds
        do⟨statement⟩
(2) case begin⟨statement⟩group of
        L1: ⟨statement⟩
        L2: ⟨statement⟩
         ⋮
        end
```

The value of a group is determined by the statements

success,
failure, or
case⟨const⟩.

The statements *success* and *failure* are used in boolean groups, and the statement *case*⟨const⟩ is used in non-boolean groups. These statements transfer control to the end of the group and assign the group the value *true* (success—succeeds, failure—fails), *false* (success—fails, failure—succeeds), or the value of ⟨const⟩.

Example:

```
begin repeat⟨statement⟩
        until begin repeat if⟨cond1⟩then success
                until⟨cond2⟩; failure
            fails
        end
```

In the compiled program, the values defined by a group would usually not materialize; instead, each branch to the end of the group would be extended to the program part to be executed next. Since transfer of control goes to the end of a group and since the group brackets can be set arbitrarily, it is possible to specify loop terminations that bridge across several loop levels.

(B) As a second structure, a new loop construct is suggested that combines the *case* and the *while* statement. Its form is

```
repeat case⟨expression⟩of
L1: ⟨statement⟩
L2: ⟨statement⟩
 ⋮
exit on
M1: ⟨statement⟩
M2: ⟨statement⟩
 ⋮
end
```

This statement executes the statement labeled by L1 (L2, L3, etc.) repeatedly while ⟨expression⟩ yields L1 (L2, L3, etc.), and it terminates the loop executing the statement labeled by M1 (M2, M3, etc.) when ⟨expression⟩ yields M1 (M2, M3, etc.).

SUMMARY

The set of basic flow structures identified in this paper permits the construction of all flow graphs that do not contain multiple-entry loops. These are the same graphs as those that can be fully reduced by Allen and Cocke's method of interval reduction. It has been pointed out that programs based on the described system have most, if not all, of the positive characteristics of Structured Programs. Further, by extending Wirth's language PASCAL, a set of new control constructs has been suggested that support the proposed set of flow structures. We might add that PASCAL extended in this way, should not anymore contain the goto-statement.

REFERENCES

1. Allen, F. E. and J. Cocke, "Graph-theoretical Constructs for Program Control Flow Analysis," *IBM Research Report RC3923*, July 1972.

2. Böhm and Jacopini, "Flow diagrams, Turing machines, and languages with only two formation rules" *CACM* 9, 5, May 1966.
3. Dijkstra, E. W., "Goto Statement Considered Harmful," Letter to the Editor, *CACM*, 11, 3, March 1968.
4. Dijkstra, E. W., *Notes on Structured Programming,* August 1969.
5. Hopkins, M. E., "A case for the GOTO," *ACM Annual Conference 1972.*
6. Knuth, Floyd, *Notes on avoiding 'GOTO' Statements,* Technical Report CS 148, Stanford University, January 1970.
7. Leavenworth, B. M., "Programming With(out) the GOTO," *ACM Annual Conference 1972.*
8. Lindsey, C. H. and S. G. Van Der Meulen, *Informal introduction to ALGOL 68,* North-Holland Publishing Co,. Amsterdam, London, 1971.
9. Mills, H., "Top down programming in large systems," *Debugging Techniques in Large Systems* (Ed. Rustin Randall) Prentice-Hall, Englewood Cliffs, N.J. 1971.
10. Wirth, N., "Program development by stepwise refinement," *CACM* 14, April 1971.
11. Wirth, N., "The programming language Pascal," *Acta Informatica,* 1, 35-63.
12. Wulf, et al., "Bliss: a language for systems programming," *CACM,* December 1971.
13. Wulf, W. A., "A case against the GOTO," *ACM Annual Conference 1972.*
14. Wegner, Eberhard, "Tree-structured programs," *CACM,* 16, 11, November 1973.
15. Kosaraju, S. Rao, "Analysis of structured programs," Proc. of 5th Annual ACM Symp. on Theory of Computing, May 1973.

Computer performance analysis in mixed on-line/batch workloads

by JO ANN LOCKETT

The Rand Corporation
Santa Monica, California

INTRODUCTION

Analysts may measure and analyze a mixed-workload system for a variety of reasons. As a first example, it may be necessary to cut costs by reducing equipment. Before arbitrarily removing some piece of equipment, management should determine how the on-line and batch systems will be affected. If problem areas can be identified prior to equipment removal, action can be taken to reduce any negative impact. The next decision may then involve the selection of equipment to remove. This decision could best be made if comparative data on the performance of the various configurations were obtainable.

A second motivation might be to determine whether the present system can accommodate more terminals. Tests could be conducted to determine the performance of the system with the current configuration and its degredation as the on-line load is increased.

A third motivation might be to determine whether a system can perform some function within a specific time interval. In this circumstance it is important to be able to identify the maximum load (or number of terminals) the system can support and still be able to perform the critical functions.

As a final example, an installation may need to identify the specific causes of saturation. If the system resources which contribute most strongly to saturation are known, management can limit the use of those elements by several means including, (1) raising the cost of using the resource, (2) making it difficult for the user to obtain the resource, and (3) scheduling the use of the resource. With this kind of information adverse situations can be avoided.

The importance of computer performance evaluation has been recognized for some time.[1,2,3] However, the most meaningful measurements for analysis and evaluation of batch systems are often difficult to obtain. Many problems occur in measurement of multiprogramming and multiprocessing systems, usually because of the complexity of current operating systems. Ideally, measurements are conducted in a controlled environment, but this approach of total control is not always feasible due to critical processing requirements or simple cost considerations. Furthermore, results from a controlled environment may be applicable to the artificial loading situation only. Data gathered from a system with a realistic workload is often necessary. Even more problems surface in on-line and mixed-workload systems. More complexities are imposed on the system as services such as swapping and time-slicing are introduced. In a batch shop, scheduling can be controlled by the operator; in a mixed-workload environment most of the control is lost and loads are generated randomly. The need for realistic performance data, however, is at least as large in this mixed-workload environment as in the simpler batch-only or on-line only systems.

REDUCING MEMORY AT RAND

The techniques described in this paper were tested on a specific, real system. They appear generally applicable to a wide variety of situations like those previously described.

At The Rand Corporation the problem of measuring a mixed workload was addressed when one of the on-line systems, Conversational Programming System (CPS), was receiving very little use. CPS required 648K bytes of dedicated core storage on an IBM 360/65. The Rand Computation Center was interested in reducing costs and therefore investigated the removal of CPS and one megabyte of bulk storage from the system. This reduction of resources would be performed only if users would not be adversely affected by the reduction of resources.

Reports showed that CPS use at Rand had been low for some time. Many changes had occurred both in the Rand workload and in the system since CPS had initially been introduced. One addition to the system, WYLBUR,[4] is a text editor with remote job entry capabilities. Many CPS users had converted to WYLBUR. Because CPS usage was so low, questions were raised regarding the removal of CPS and its supporting (and costly) resources from the system.

CPS required the dedicated use of 648K bytes of core storage on the IBM 360/65 operating OS/MVT. To support CPS, we leased one megabyte of Ampex Extended Core Memory (ECM). The remaining 376K bytes were used for other general purposes. Determining the functional effects of removing the portion of core dedicated to CPS was simple; CPS users were asked whether CPS removal would adversely impact their research. Determining the performance effect of removing the remaining core was not so simple; we had no idea of the role it played in the system. There would be less core for batch jobs so many functions would be running in slower, IBM-supplied Large Core Storage (LCS).

The system at the time included an IBM 360/65 running OS/MVT. The system was configured with a good deal of foreign (non-IMB) equipment. Storage consisted of 512K bytes of highspeed core, one megabyte of IBM Large Core Storage (LCS), and one megabyte of Ampex Extended Core Memory (ECM). Analyses were conducted assuming that if core were returned, it would be the Ampex core. This was for two reasons. First, due to certain lease arrangements, it appeared to be infeasible to return the LCS. Secondly, by assuming a worst case for the study (keeping the slowest core), we would not have to repeat the tests if the LCS could be returned.

TEST DESIGN

The most effective way of determining the effects of a modification is often to try it. In this case we could try the proposed system modification by removing memory. The AMPEX ECM was not physically removed for the tests. Since IBM machines are able to run with less core than is actually present, a simulated removal of the ECM was achieved by loading the system (through an IPL) to run with only one and one-half megabytes of storage. In this way the ECM, which resided in the highest locations, was effectively removed.

Data were collected for a three-week period. The full two and one-half megabyte configuration was used during the first and third weeks. The proposed one and one-half megabyte configuration was used during the second week. The first week was treated as a control week but its purpose was to insure that the tests were adequate, that the procedures and data collection did not adversely affect either the users or the system, and to allow time to solve any problems which might be encountered prior to the actual test weeks. The second week was the true test week during which time the smaller configuration was run. The third week was the control week. During this week data were collected on the normal configuration. Data collected during this week would provide a basis for comparison with data collected during the test week.

To evaluate the effect to the system of removing CPS and one megabyte of bulk core, we first had to identify the areas of performance by which the service of the computer could be judged. Then we needed specific metrics and decision criteria to specify acceptable levels of service prior to the evaluation. Otherwise we could fall into the trap of collecting data without purpose and subsequently making an arbitrary decision. We felt that four items, each likely to be affected by the removal of the ECM, would reflect the general level of service provided. The four user-related variables were selected because we were specifically trying to determine the effect of the change on the users. The four items were

- CPU utilization
- batch turnaround
- WYLBUR response time,
- BIOMOD availability

To assure uniform collection of data, forms were designed and reproduced for use in the on-line tests. We found forms to be quite useful because both BIOMOD and WYLBUR tests were conducted by several people. The forms assured that the same data were collected each time.

CPU UTILIZATION

Software monitors are tools frequently used to evaluate computer performance. The Boole and Babbage monitor, Computer Utilization Evaluator (CUE) can provide a large amount of data, depending on user selected parameters. CUE was run for an hour, twice each day for the three-week test period. Although overhead required to run CUE is not prohibitive, parameters were selected to furnish required data with minimum impact to the system. This was to maintain an environment for testing as close to normal as possible. Data on CPU utilization and device and channel activity were collected at half-second intervals. CUE was always run in 14K bytes of high-speed storage.

We measured CPU utilization because the proposed system configuration had the master scheduler, linkpack area, and readers residing in slower core. These modules, executing in slower core, could increase CPU utilization to saturation. A saturated processor would limit future system modifications, so any tendency toward this condition should be detected during the test. Elimination (or severe reduction) of excess capacity could be adequate reasons for rejection of the core reduction; our decision criterion was to allow a 50 percent reduction in idle time of the processor.

BATCH TURNAROUND

Normal batch turnaround for jobs run during the day had averaged 30 minutes for quite some time. Users queried regarding acceptable limits of turnaround seemed to willingly accept batch turnaround up to one hour without feeling inconvenienced; extension of turnaround time beyond one hour would constitute an unacceptable situation. (However, we were only concerned with jobs run during the day shift. Priority was given during the day to small jobs requiring 64K to 104K bytes of storage and executing in less than 200 CPU seconds. Turnaround during the night shift was not considered critical since most users waited until morning to get their output.)

WYLBUR

WYLBUR is an on-line text-editor and has remote job entry functions similar to those performed by the remote job entry functions of CPS. WYLBUR is used to create, store, retrieve, and modify text. This is by far the most popular of the online systems at Rand and is used for many things including preparation of reports, entry to the ARPA network of computers, and the coding and execution of computer programs. Due to its popularity, there is concern for maintaining both reliability and acceptable response time of the system.

TABLE I—CPU Utilization

WEEK 1

DAY	TIME	CPU UTILIZATION (Percent)
Monday	3:23- 4:27 PM	86.63
Tuesday	9:07- 9:42 AM	50.97
	2:14- 3:16 PM	45.75
Wednesday	10:09-11:10 AM	65.81
Thursday	2:27- 3:27 PM	47.33
	4:37- 5:38 PM	61.47
Friday	9:31-10:32 AM	72.54
	1:30- 2:30 PM	62.68

MEAN CPU UTILIZATION = 61.65 Percent
STANDARD DEVIATION = 12.91 Percent

WEEK 2

DAY	TIME	CPU UTILIZATION (Percent)
Monday	10:56-11:58 AM	73.89
	3:20- 4:20 PM	73.64
Tuesday	2:11- 3:11 PM	55.65
Wednesday	10:02-11:03 AM	73.10
	3:26- 4:26 PM	81.51
Thursday	2:27- 3:27 PM	69.53
	4:27- 5:28 PM	62.83
Friday	9:42-10:43 AM	64.64
	1:28- 2:29 PM	58.17

MEAN CPU UTILIZATION = 68.11 Percent
STANDARD DEVIATION = 8.40 Percent

WEEK 3

DAY	TIME	CPU UTILIZATION (Percent)
Monday	10:42-11:42 AM	72.74
	3:04- 4:04 PM	86.58
Tuesday	2:01- 3:01 PM	53.72
	4:03- 5:03 PM	78.53
Wednesday	9:56-10:56 AM	51.44
	3:28- 4:28 PM	57.64
Thursday	2:27- 3:27 PM	62.22
	4:22- 5:22 PM	53.92
Friday	9:30-10:30 AM	64.08
	1:37- 2:37 PM	67.22

MEAN CPU UTILIZATION = 64.81 Percent
STANDARD DEVIATION = 11.00 Percent

Because of the heavy use made of WYLBUR, it was important to determine the effect of the planned system modification on it. The competition of higher-priority tasks (including I/O operations) running in slow core could adversely affect WYLBUR response time. Experience indicates a person using WYLBUR will be intolerant of consistently long response times for frequently used commands and of frequent lockouts (excessive response times). If WYLBUR were adversely effected, removal of the core would not be considered. We quantified this into the criterion that a difference in mean response times for the tests and the control weeks of more than twice the pooled standard deviation would constitute an unacceptable situation.

BIOMOD

BIOMOD is an on-line system with full vector graphics capabilities.[5] It is a highly interactive system and, due to its flexible, model structuring capabilities, can be very effective in studying models of dynamic systems. BIOMOD utilizes a graphics console composed of a television screen, data tablet, and keyboard and is active for fairly long periods of time (15 minutes to several hours). It requires 228K bytes of core and is usually run in a fixed partition in ECM since running in high-speed core would interfere with the ability of batch users even to obtain core space. BIOMOD's real-time recognition of hand printed requires rapid processing of tablet data, and this speed can be achieved more easily in ECM than LCS. Running BIOMOD in the slower LCS could result in too much tablet data being lost, and the system would become useless. The criteria for BIOMOD was that the general performance not be severely degraded.

A test was constructed to evaluate both the interactive and non-interactive features of BIOMOD. The testing of BIOMOD was the weakest part of the evaluation as a result of including subjective measures in the test because the actual performance of BIOMOD under the proposed configuration was not important as long as it remained usable.

SPECIAL ON-LINE TESTING PROCEDURES

A session on WYLBUR usually involves the use of several types of commands. These commands can be categorized as text input (collect command), file activity (use and save commands), and text editing (modify, change, and align commands). To measure WYLBUR response, it is not necessary to emulate a "typical script" but rather to measure response of each of the various commands which are used to make up a script. The commands selected for measurement were those most frequently used. To ensure that each iteration of the test would be identical, permanent test data files were created and the test designed so that input data sets were never disturbed.

A stopwatch was used to clock the time between the message being sent and the response. Part of the on-line testing employed artificial stimulation of the system, using a special

TABLE II—Turnaround Statistics

Average Turnaround Time (hours)	All Classes	L	A	B	Other*
Week 1					
Mean	0.51	0.28	0.43	0.67	0.68
Standard deviation	0.078	0.029	0.046	0.157	0.164
Week 2 (without ECM)					
Mean	0.72	0.50	0.58	1.11	0.48
Standard deviation	0.092	0.129	0.055	0.183	0.695
Week 3					
Mean	0.54	0.33	0.48	0.80	0.95
Standard deviation	0.16	0.052	0.12	0.40	0.44
Number of Jobs Completed					
Week 1					
Mean	188.75	17.50	95.50	64.00	11.75
Standard deviation	11.52	7.76	10.87	6.67	3.90
Week 2 (without ECM)					
Mean	188.75	18.25	112.00	56.00	2.50
Standard deviation	15.66	7.50	14.88	4.64	3.28
Week 3					
Mean	203.75	28.25	116.75	48.25	10.50
Standard deviation	17.37	4.82	12.83	12.26	2.29

* Job class determines job priority and is used to provide balance between I/O and CPU jobs. Maximum region size for class L jobs is 64K. Class A jobs may run in 65-104K and class B in 105-228K. All three classes are restricted to 150 pages, 500 punched cards, and a total of 10,000 I/O operations.

hardware device. (This device has been previously referred to as a surrogate keyboard.[6]) The hardware device was used to transmit commands to the IBM 360/65 at maximum rate; a stopwatch was used to measure the time between a message being sent from a terminal (carriage return) and the returned response. Each command of part 1 of the WYLBUR test was processed five times. The collect command was tested in part two of the test. We recorded the number of lines collected in each of thirty-six five-second intervals. WYLBUR system status information was recorded several times during the test. The tests, each requiring 40 to 50 minutes, were run twice each day during the system's busiest hours. The WYLBUR tests were conducted by several people since they proved to be rather tedious.

The BIOMOD tests were conducted by three BIOMOD users. These tests were run each day during the control and test weeks. Tablet performance was measured by counting unrecognized characters and by subjective evaluation of certain specified vector-graphic operations. Elapsed times required for model retrieval and simulation, and for construction of displays, was used to evaluate the non-interactive features of BIOMOD.

TEST RESULTS

The CPU utilization determined by CUE runs during the three-week test period is shown in Table I. Each run was scheduled for an hour during one of the system's heavy-use intervals (9:30-11:30 a.m. and 1:30-5:30 p.m.). Various operational problems prevented some runs from being made according to schedule. The difference between CPU utilization with and without ECM was not significantly different.

The average turnaround time for the various job classes is summarized by week in Table II. Turnaround for this report is defined as the elapsed time between a job being submitted until the output is available. The average turnaround time for jobs run during the prime shift (8:00 a.m. to 6:00 p.m.) increased from 30 minutes to 43 minutes when ECM was removed. Since we had previously decided to allow average turnaround time to increase up to one hour, the data showed that we could remove ECM and still meet our performance goals for turnaround time.

A summary of WYLBUR response times for the various commands tested is shown in Table III. The data do not show any correlation between response time and the removal of storage. Examination of individual tests showed more correlation with the time of day and the number of WYLBUR users than with the presence of ECM.

Degradation did occur in BIOMOD as shown in Table IV. Times required for model retrieval, display construction, and plotting increased significantly. Pen data was not significantly affected, although some pen up/down data was lost when ECM was not used. However, the system did remain usable.

CONCLUSIONS

The test procedures allowed us to determine the effects of the proposed system modification. The results showed we could easily remove CPS and one megabyte (either LCS or

TABLE III—WYLBUR Response Time

	Week 1	Week 2 (without ECM)	Week 3
USE			
Mean	11.50	13.49	14.14
Standard deviation	1.17	1.47	2.56
Average number of users	3.6	4.0	5.5
ALIGN			
Mean	48.78	64.34	70.35
Standard deviation	4.70	11.09	16.02
Average number of users	3.6	4.0	5.5
SAVE			
Mean	27.48	32.63	34.86
Standard deviation	2.62	4.14	4.76
Average number of users	3.6	4.0	5.5
CHANGE (with replacement)			
Mean	29.18	27.90	29.97
Standard deviation	7.01	4.12	5.83
Average number of users	4.05	4.3	5.5
CHANGE (scan only)			
Mean	23.12	20.51	22.98
Standard deviation	10.57	3.20	3.37
Average number of users	4.05	4.55	5.25
LIST			
Mean	46.40	51.23	56.98
Standard deviation	8.52	11.89	17.53
Average number of users	4.15	4.2	4.95
COLLECT			
Mean[1]	16.63	17.09	17.96
Standard deviation	3.27	2.05	1.72
Average number of users	4.2	4.05	4.4
COLLECT			
Mean[2]	18.34	16.91	17.94
Standard deviation	2.34	1.40	1.02
Average number of users	3.9	4.4	4.0
COLLECT			
Mean[3]	18.89	16.93	16.45
Standard Deviation	1.63	2.26	2.15
Average number of users	4.2	4.3	4.0

TABLE IV—Summary of BIOMOD Test

	Week 1 (with Ampex)	Week 2 (without Ampex)
Misrecognized characters		
Mean	4.18	4.67
Standard deviation	1.27	2.06
Pen-tracking grade*		
Mean	2.77	2.47
Standard deviation	0.88	0.74
Model Retrieval Time		
Mean	13.09 sec.	17.83 sec.
Standard deviation	2.64 sec.	3.02 sec.
Display Construction Time		
Mean	6.00 sec.	12.17 sec.
Standard deviation	1.28 sec.	1.46 sec.
Construction-Phase Grade*		
Mean	3.03	2.02
Standard deviation	0.60	1.01
Plot Time		
Mean	14.75 sec.	22.50 sec.
Standard deviation	7.14	3.86
Simulation-Phase Grade*		
Mean	2.87	2.40
Standard deviation	0.52	0.52
Overall System Grade*		
Mean	3.01	2.06
Standard deviation	0.406	0.98

* These values were obtained by converting subjectively assigned grades to the values A=4, B=3, C=2, D=1, and F=0.

ECM) without causing a major impact to our operations. The approach used for measuring the on-line systems was very simple and direct, but provided some insights into the problem of analyzing computers with combined on-line and batch systems. The suggestions below describe some of these insights that appear applicable to most such computer performance analyses.

- Complete test planning and pilot testing are extremely valuable. Allowing extra time, prior to actual testing, to try out the testing and data collection procedures will assure that tests provide adequate and meaningful data on which to base important decisions.
- Metrics should be chosen prior to test design. Tests can can then be structured to gather information specifically needed for analysis and to avoid collection of unnecessary data. Metrics should reflect the goals of the evaluation. For example, user-directed metrics should be employed if the intent is to measure the impact on users.
- Decision criteria should be chosen prior to data collection. The data can be collected in a form which will reduce time spent on analysis. Prior definition of these criteria force the analyst into defining his goals without being biased by the results of data collection.
- Useful measurement of on-line systems is possible and can be done without expensive equipment.

We found that although simple measurements can be useful, we need a method by which large amounts of data can be collected. Since many responses are highly sensitive to the system load and number of users, large sample sizes are required for analysis. A different method of collecting data may be appropriate for use in analyzing on-line systems. It appears that "average" response time is not a meaningful measure. Thresholds may exist for which the actual values of outliers are not meaningful. For example, when using a specific command, a person at a terminal may not be able to distinguish between a response time of .01 second and another of .1 second. He may also require enough "think time" that a response less than one minute is sufficient and system improvement to achieve a better response time would be wasted. At the other extreme, a person may tolerate bad response, but only to a point. If response is continually too slow, the user will stop using the terminal. The relative response times are not important if they are too slow. The important data are the responses that users desire and those they will tolerate. A performance goal of an on-line system should be to consistently maintain response time between the two critical thresholds. Mean response time, which will include outlying values does not provide an adequate measure. A better measure may be simple counts of the number of responses between and on either side of the thresholds.

REFERENCES

1. Boehm, B. W., *Computer Systems Analysis Methodology: Studies in Measuring, Evaluating, and Simulating Computer Systems*, The Rand Corporation, R-520-NASA, September 1970.
2. Bell, T. E., B. W. Boehm, and R. A. Watson, "Framework and Initial Phases for Computer Performance Improvement," *AFIPS Conference Proceedings*, Vol. 41, Fall Joint Computer Conference 1972, pp. 1141-1154.
3. Bell, T. E., *Computer Performance Analysis: Measurement Objectives and Tools*, The Rand Corporation, R-584-NASA/PR, February 1971.
4. Fajman, R., and J. Borgelt, "WYLBUR: An Interactive Text Editing and Remote Job Entry System," *Comm. of ACM*, Vol. 16, No. 5, May 1973, pp. 314-322.
5. Groner, G. F., R. L. Clark, R. A. Berman, and E. C. DeLand, "BIOMOD: An Interactive Computer Graphics System for Modeling," *AFIPS Conference Proceedings*, Vol. 39, Fall Joint Computer Conference 1971, pp. 369-378.
6. Bell, T. E., *Computer Performance Analysis: Minicomputer-based Hardware Monitoring*," The Rand Corporation, R-696-PR, June 1972.

Systems performance/measurements—A quantitative base for management of computer systems

by PAUL MALICK

United Air Lines
San Francisco, California

INTRODUCTION

Systems Performance/Measurements is a new area of computer technology which provides a quantitative base for management of data processing systems. This function has generated high interest due to increasing emphasis on cost effective management of these data processing systems.

The people, the tools used and results obtained are the subject of this paper.

SYSTEMS PERFORMANCE/MEASUREMENTS AT UAL

Airline industry problems

The airline industry has a unique set of problems to solve. They are:

- High Capital Investment
- Large Maintenance Requirement
- Highly Service Oriented
- Highly Regulated
- Highly Unionized
- Geographically Spread Out
- Highly Competitive Route Structure
- Stringent Security Requirements

Because of these problems, United Air Lines has been, and is, a pioneer in computing. The early systems were primarily batch; however, as the technology became available, the on-line systems were brought into being by working closely with vendors to develop the required systems. Newer, faster, more powerful and reliable systems are being developed as their need can be economically justified.

UAL computer organization

United has formed the Computer and Communication Services Division to effectively manage the large investment in resources, both machines and people.

Figure 1 shows the United Air Lines Computer Centers, equipment, major users and interaction.

United Air Lines has four computer centers, located in three cities, as well as a data communications network serving all cities throughout the airline's system and connected to other airlines. Non-polled networks or links connect the four centers. These centers serve a diverse group of major users, including: Finance, Marketing, Maintenance and Operations. There are a large group of medium and smaller sized users including Corporate Planning, Food Services, Personnel, Industrial Engineering, etc.

Planning/measurements cycle

Effective management of a large capital investment in computer systems requires a quantitative base which must relate to the division planning. Figure 2 depicts the planning/measurements cycle which relates system performance to division planning.

Resource measurements indicate where we are now. The measurements are used as:

- Input to Planning
- The basis of chargeback to users for resources used.

The resource areas measured are: central site hardware, application software, systems software and network/remote hardware.

The plan indicates where we are going and when we will get there. Plans can be separated into three time periods as follows:

- Near term—days, weeks
- Short term—up to a year
- Long term—greater than a year

The near term problem is usually solved using measurements directly. Modeling and simulation using measurements as input are often required to provide the quantitative base of the plan for the short and long term.

A very important part of the planning process is to compare the planned vs. actual and feed the result back into the plan. The objectives of the feedback are to highlight mistakes so they can be corrected and serve as a basis for improving the planning mechanism.

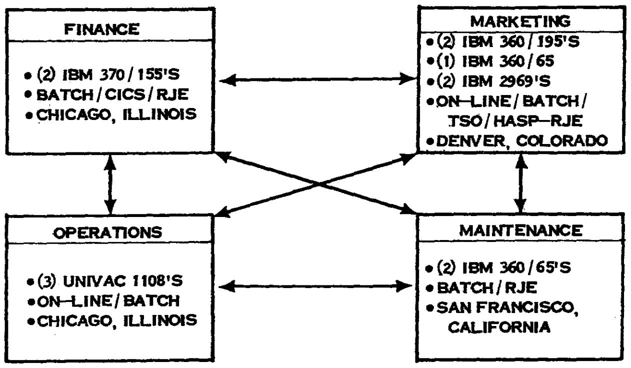


Figure 1—United Air Lines computer centers, equipment, major users and interaction

Systems performance/measurements groups

The Systems Performance Measurement groups at United Air Lines provide most of the quantitative base for management of the computer systems. The groups answer the following basic questions:

1. How much of and how well are we utilizing the current resources, quantitatively?
2. What impact will hardware, software and applications changes and/or additions have on systems resource utilization, quantitatively?

The answering of these questions requires the use of the appropriate tools. In the case of question number one, the tools would primarily be software and hardware monitors. The tools for answering question two are a combination of software and hardware monitoring as well as simulation and modeling.

The results of the use of the tools along with analysis, are systems performance reports and recommendations. The report with the largest impact is the "Planning and Capacity Report" for each resource center. This report is used by division management as the prime quantitative input to planning.

The personnel in the Systems Performance area are all experienced computer people and tend toward the senior level. Each tends to have a systems viewpoint, be highly analytical, and work well with various people.

Planning and capacity reports

A planning and capacity report for a particular computer center contains the following:

1. Resource utilization, by resource, for each of the next five years. The on-line system resource utilization is broken out by user application.
2. Prediction of system bind points by resource and year.

All resource utilizations include changes and additions of hardware, software and applications at the time predicted.

RESOURCES MANAGED AND TOOLS USED

Several tools are used to provide the quantitative base for management of resources. The tools are measurement, simulation and modeling. The tools provide data for answering the two basic systems performance questions.

Resources managed

Figure 3 depicts the five computer resource areas to manage. These areas are managed as a system as opposed to separate entities. It should be noted, an on-line system contains all five areas while a batch system does not have a data communications network to manage. It is interesting to note that an on-line system is primarily self-scheduled and a batch system is primarily manually scheduled.

The central processing unit executes instructions, thus the most important parameters are:

- CPU activity (which is broken into):
  —operating system
  —application or problem program
- Average Instruction Execution Time (AIET)

The memory module holds programs and data. The prime parameters are:

- Memory Utilization (by whom)
- Memory Contention (multi-processor systems only)
- High Speed Buffer Hits/Misses

The Input/Output channels transfer data to and from peripheral devices and memory. The prime parameters are:

- Channel Utilization (peripherals)
- Channel Utilization (computer-to-computer)

The peripheral devices input, output and/or store data. The major parameters are:

- Control Unit Utilization
- Device Utilization
- Average Access Time
- Number of Accesses

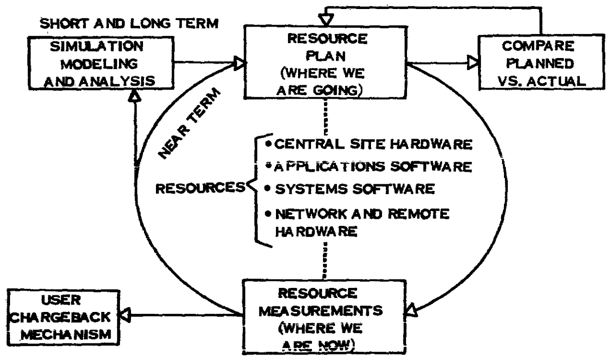

SHORT AND LONG TERM
SIMULATION MODELING AND ANALYSIS
RESOURCE PLAN (WHERE WE ARE GOING)
COMPARE PLANNED VS. ACTUAL
NEAR TERM
RESOURCES
• CENTRAL SITE HARDWARE
• APPLICATIONS SOFTWARE
• SYSTEMS SOFTWARE
• NETWORK AND REMOTE HARDWARE
USER CHARGEBACK MECHANISM
RESOURCE MEASUREMENTS (WHERE WE ARE NOW)

Figure 2—Planning/measurements cycle

The data communications network provides a data path to and from devices which are remote from the computer. The network can be envisioned as a peripheral device which is more complicated than any other. The prime parameters are:

- Data Communications Line Loading
- Number of Messages
- Number of Characters Per Message

There are various system parameters which are important because they provide data which show quantitatively the relationships between the parts. These are:

- Number of file accesses
- Queue levels
- Job/Step Log
- Number of concurrently active channels
- Number of jobs or transactions concurrently active.
- Amount of time instruction execution is overlapped with channel activity.
- Job transaction elapsed time
- Remote device response time

System optimization is concerned with the "balance" of all the parameters identified.

Measurements

Basic questions answered by measurements

There are three basic questions answered by measurements:

a. What percent of the resources are being used now?
b. Who is using what percent of the resources?
c. What are the specific device characteristics for the system being measured (AIET, average access time, etc.)?

The answer to the first question indicates the capacity of the resource being used, thus the reserve capacity can be calculated. The answer to the second question indicates who is consuming the resources. The data obtained by answering the third question is used as input to the simulation/modeling process. The measurement tools used are hardware and software monitoring.

Hardware monitoring

Unique characteristics

A hardware monitor has a set of unique characteristics as a measurement tool:

a. It is transparent to (no-load on) the system being measured.
b. It can capture 100 percent of the samples.
c. It captures data not available by any other method (such as Average Access Time, Average Instruction Execution Time, etc.)

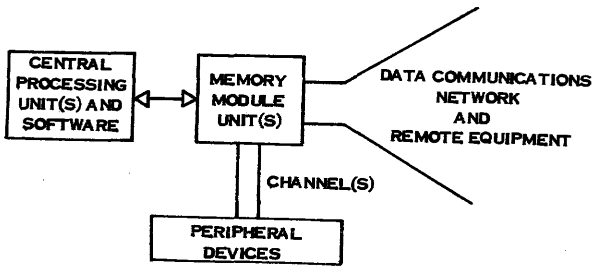

CENTRAL PROCESSING UNIT(S) AND SOFTWARE
MEMORY MODULE UNIT(S)
DATA COMMUNICATIONS NETWORK AND REMOTE EQUIPMENT
CHANNEL(S)
PERIPHERAL DEVICES

Figure 3—Five computer resource areas to manage

The first characteristic indicates that no distortion is placed upon the system measurements. The second characteristic denotes the degree of confidence that the measurements taken are representative of the period measured. The third characteristic means that the key data required for modeling is available and accurate.

Choosing a hardware monitor

United Air Lines owns a triplex Comress D-7900 hardware monitor system. The choice of this particular system was made based upon the following prime characteristics.

a. Number of Counters—determines how many functions or events can be accumulated separately (example: monitoring all storage activity).
b. Plugboard Logic Capacity—determines how much signal combination and translation can take place simultaneously (example: core utilization).
c. Maximum Counting Rate—the counters should count at a rate equal to the speed of the signal being measured so 100 percent of the signals can be captured. Note: this can sometimes be accomplished using plugboard logic. However, the logic is then unavailable for other purposes and plugboard logic is a critical resource.
d. Built-in Signal Simulator—is used to check out plugboard logic before the monitor is connected to the system. Without this capability, checkout of plugboard logic is difficult, time-consuming and often inaccurate. This feature is a must!
e. Modular, Reliable and shipped easily—the system must be capable of being broken into smaller elements for shipping. Modularity helps this situation greatly. Reliable units are a must to develop confidence in the results.

Using a hardware monitor

The use of a hardware monitor follows a very definite pattern which is shown in Figure 4, Hardware Monitor Use Flow.

The first step is to set up a test plan. The question to ask is: "What are the results I want to obtain?" Once the ob-

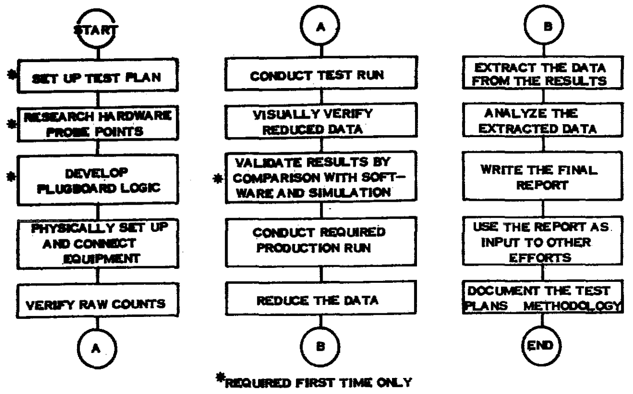

START
* SET UP TEST PLAN
* RESEARCH HARDWARE PROBE POINTS
* DEVELOP PLUGBOARD LOGIC
PHYSICALLY SET UP AND CONNECT EQUIPMENT
VERIFY RAW COUNTS
A
A
CONDUCT TEST RUN
VISUALLY VERIFY REDUCED DATA
* VALIDATE RESULTS BY COMPARISON WITH SOFTWARE AND SIMULATION
CONDUCT REQUIRED PRODUCTION RUN
REDUCE THE DATA
B
B
EXTRACT THE DATA FROM THE RESULTS
ANALYZE THE EXTRACTED DATA
WRITE THE FINAL REPORT
USE THE REPORT AS INPUT TO OTHER EFFORTS
DOCUMENT THE TEST PLANS METHODOLOGY
END
*REQUIRED FIRST TIME ONLY

Figure 4—Hardware monitor use flow

jective is established, the necessary effort can be expended to plan toward and reach that objective.

Researching the probe points has been necessary in many cases because the probe points are not available in any probe point library. This usually doesn't reflect probe point library inadequacy, only that the problem probably hasn't been solved before. Most of the standard measurement probe points are in the library, however. It is important to document the test plan, probe points and logic so they can be used again with a minimum of effort.

Validation of the resulting data with whatever other data available is necessary to develop confidence in the results. A direct comparison with software measurements is ideal. However, when these are not available, trends and relationships with other measurements or simulation can be used.

Writing a final report solves several problems. First, an historical record is available to indicate the results whenever they are needed. Second, the report provides the starting point for the next time the tests are run. Finally, organizing the results into report form often uncovers problems with the tests which can either be corrected immediately or the next time the test is run.

Measurements obtained using the hardware monitor

Hardware Monitor measurements obtained include the following:

- CPU Utilization
- Channel Utilization
- Core Utilization
- Memory Contention (on a multi-processor system)
- High Speed Buffer Hits
- Computer to Computer Channel Utilization
- Average Instruction Execution Time (AIET)
- Peripheral & Peripheral Control Unit Utilization (Disk, Drum, Printer, Tape, etc)
- Drum & Disk Average Access Time
- Data Communications Network Characters transferred.

These measurements were obtained on various United Air Lines computer systems including: IBM 360/50, 65, 370/155 and UNIVAC 1108, plus an IBM 2969 (Programmed Terminal Interface). It should be noted that most of these measurements would be of equal interest for either an on-line or batch system. Only the values and their relationships change.

Hardware monitor measurements represent a composite of all individual times or events. Thus, another measurement method is normally required to reconstruct the components which make up the total. This method is software monitoring.

Software monitoring

Unique characteristics

A software monitor has a set of unique characteristics as a measurement tool:

a. The counter mechanism is connected directly into the system software thus the counter and collection mechanism bias the results.
b. Data collection is done on a sampling basis because of the bias mentioned above.
c. It captures only data available internally in the computer.

The last characteristic means that this measurement tool can be used to capture transaction, program, job and file data. Software monitors are normally continuously resident in the computer. Thus, they can be turned on at will for diagnostic or other purposes without delay. It should be noted the sample rate must be carefully chosen to balance the mechanism bias on the results with the validity of the sample.

Software monitors used

The following software monitors are used at United Air Lines.

a. Systems Management Facility (SMF)—Is a standard IBM OS system measurement package. In addition to standard data reduction, Mark IV file management package (from Informatics) is used to easily reduce the data for special purposes.
b. MFKEYS—Is a data collection and reduction package developed by UAL used to capture CPU and drum subsystem utilization as well as key system parameters from the operational computer system.
c. Apollo Data Collection and Reduction—Is a UAL developed package used to measure the Apollo reservation system utilization. The measurements include CPU, File, Message Mix, Program and transaction activities.
d. Compool Snap—Is a UAL developed data collection and reduction package for the operational computer system which is used to capture message and program hand-off data.

e. PTI Data Collection—Is a vendor developed data collection and reduction program used to collect core utilization, data communication line and message data in the IBM 2969 Programmed Terminal Interface system for the Apollo reservations system.

Each software data collection package is used for a specific area as identified. It is necessary to have tools to quantify the activity in each resource area so a full picture can be developed.

Measurements obtained using software monitors

The following measurements have been obtained using software measurements:

- CPU Utilization (total, by job or transaction)
- Channel Utilization
- Core Utilization
- Peripheral Unit Utilization
- File accesses by File (Data Set) by Job and Step
- Job and Job Step Log, Start, End
- Number of Operating System Requests
- Queue Levels
- Number of Transactions by Transaction Type
- Program/Job/Step Execution Time
- # Messages
- Length of Message

These measurements were obtained on various United Air Lines computer systems as previously noted under hardware monitoring.

Simulation/modeling

Basic questions answered by simulation & modeling

Simulation and Modeling are the tools used to predict the future in terms of resources. The basic questions answered are:

1. What impact will various changes and/or additions have on the system quantitatively?
2. When and at what level will the system saturate?
3. What are the system bind points?

The answers to these questions become the basis for a Planning and Capacity Report. It should be noted that correct predictions are made on the basis of an accurate picture of the present. This accurate picture of the present is based upon actual measurements.

Modeling description

There are two parts to the modeling process. They are the model mechanism and the model input.

The model mechanism represents the system being modeled. The total computer system model consists of the system software (operating system), the system hardware and their interaction. The dynamic interaction of these parts are simulated by the use of various algorithms and probability distributions within a logical flow of events.

The model inputs provide the data base to be operated upon and the stimulus for the model mechanism. For example, the total system model inputs are the descriptions of applications, programs, files and messages along with the time of day and frequency the jobs or transactions are run.

Validation of the model mechanism and inputs are necessary both individually and as a whole. The objective is to increase the confidence in the results so they can become accepted and relied upon. The results are only as good as that which goes into the modeling process. Validation, where possible, should be with actual data.

Models have variable inaccuracies because:

- The model is not a complete representation of the system (the system itself is).
- The mechanism may have inaccuracies.
- The inputs may be incomplete and/or incorrect.

Thus, a model output which is correct to ± 5 percent is often considered very accurate.

Modeling is a time and energy consuming task. Thus, it is not always justified economically.

Models used

Where possible, existing models are used either directly or with modifications. Normally, however, an existing model has not been available. Therefore, United Air Lines has made a considerable investment in building models.

The following are the primary models used at United Air Lines:

a. Systems Analysis Machine (SAM)—Is a model developed by Applied Data Research, primarily for IBM batch systems; however, any system may be modeled. The vendor has made a large addition to the model for United Air Lines, which allows modeling of a job stream without detailed descriptions of each job and job step. This feature is a must to obtain data for preparation of a Planning and Capacity Report. The outputs from the model are CPU, channel, device and program activity statistics. Much of the detailed testing and validation of the model addition was performed by United Air Lines.

b. Operations System Planning Model—Is a model, developed by United Air Lines, of the Operational Computer System which is used to obtain the basic Planning and Capacity Report data including CPU, channel and message activity. This model is written in GPSS.

c. United Systems Simulation Model (USSM)—Is a model developed by United Air Lines in GPSS, which is used for obtaining the basic Apollo Reservations System Planning and Capacity Report data. Thus, it predicts CPU, channel and message activity.

The following special purpose models have been developed:

- Apollo Network Model
- Operational Computer Network Model
- Average Instruction Execution Time (AIET) Model
- IBM 3330 Disk Model
- Programmed Terminal Interface System Model

Each model serves or has served a specific purpose. The network models are used for data communications network management. The AIET model was used to verify the operational computer system average instruction execution time. AIET is a key input to any system model and the Hardware Monitor was not available to capture the actual value at the time. The IBM 3330 Disk Model was developed to simulate the operation of 3330's on the Apollo reservation computer system.

Special purpose models are developed as the need arises to quantify the dynamic impact of system changes before they are approved and implemented.

The magnitude of the expenditures, the possible impact upon service to our users and the risk of implementation are all factors in any decision to build a particular model.

USE OF TOOLS

The most significant and important use of the Systems Performance tools has been to provide a quantitative base for decision-making. Decisions have been made, using the quantitative base, on major changes to each of the four computing centers.

Operational computer system

The decision was made to postpone the acquisition of one million dollars of mass storage drums for a minimum of two years based upon actual measurements of average access time. The actual average access time was found, with the hardware monitor, to be 30 percent less than the manufacturer's specification. The difference was due to the amount of storage used and the data placement on the drums for the UAL application mix.

Marketing (reservations) computer system

The decision was made to upgrade to the largest capacity computer available based upon predicted resource utilization. The predictions showed the life of the new computer to be approximately five years. Thus, a costly interim conversion was avoided.

A decision was made, also, to sign contracts for computer services with several outside firms due to the availability of predicted excess capacity for the length of the contracts. Thus, revenue is being produced by using the excess capacity.

Financial computer system

The decision was made to not upgrade from a pair of 370/155 to a pair of 370/168 computers to meet Summer of 1973 peak resource requirements. Measurements indicated the peak CPU load build up was very high. Using these measurements as a base, it was predicted that the peak load would be at or near system saturation, causing great disservice to our users. System optimization and offloading of jobs to the back-up computer for the marketing system reduced the peak CPU activity by ten percentage points below the predicted peak. Thus, considerable investment in hardware and manpower were postponed for at least a year.

Maintenance computer system

The decision was made to convert this computer center to primarily a remote job entry (RJE) system, based upon predicted resource requirements and the availability of adequate capacity in the back-up computer for the marketing system. Expansion of this system will not be necessary for at least a year.

It is felt, at United Air Lines, the Systems Performance groups continuously pay for themselves by providing a quantitative base to:

- Make correct decisions
- Avoid costly mistakes

Direct cost savings, such as cost avoidance or postponement, can be calculated. Indirect cost savings, such as lost time due to mistakes, are difficult to measure.

It must be remembered that people must ultimately make the final decisions, but a quantitative base can, if used correctly, greatly assist the process.

ACKNOWLEDGMENTS

I wish to acknowledge the contributions of my fellow Systems Performance people at United Air Lines, whose efforts produced many of the tools and techniques shown in this paper. I also wish to thank the persons who reviewed this paper and made various helpful suggestions and, finally, I wish to thank United Air Lines for the opportunity to prepare and present this paper.

REFERENCES

1. *Comress Dynaprobe—7900 Users Manual*, Comress, Inc., Two Research Court, Rockville, Maryland 20850.
2. Olson, R. H. and W. T. White, *Unimatic Planning and Capacity Report*, Internal United Air Lines Report, January 1973.
3. Helm, H. L., *Financial System Planning and Capacity Report*, Internal United Airlines Report, March 1973.
4. Malick, P. D., *IBM 360/370 System Hardware Performance Monitoring and Analysis*, Internal United Air Lines Paper, January 31, 1972.

Two hat management—Project management with a difference

by ROGER W. KLEFFMAN

United Air Lines
Englewood, Colorado

INTRODUCTION

The on-line systems of the sixties were marked by an exuberant and dynamic environment of continual system implementations. New hardware, new operating systems, application growth outstripping system capacity—all led to large projects with dedicated manpower and hardware resources trying to achieve superhuman development schedules. The seventies are a different environment. New hardware and operating systems are introduced, providing increased capacity not only for application growth, but also for implementing new applications within the system.

The new environment was present at United Air Lines after cutover of an on-line airline reservation system in early 1971. The software was stable, and there was excess system capacity for new applications. All system and application program segments, on-line and off-line, were designated as specific responsibilities of first line program managers, who in turn managed programmer/analysts. There were no additional programming resources in any staff areas—the line had responsibility and accountability for all programming work effort.

Within this organization structure, three types of work would be done; correction of software errors, modification to existing coded functions, and major enhancements or development of new software products. The latter item would be carried out by means of projects. The projects would be of temporary duration; lasting only long enough to implement the new software. Upper management foresaw several problems:

- Dedicated resources for a project would be more costly than utilizing line talent on an 'as needed' basis.
- Illogical lines of management when programmers work for some period of time for a project manager, then revert back to the line manager for performance review.
- The need for technical personnel development as the old, reliable and talented people are committed to the choice projects, while new people are relegated to maintenance assignments.
- The requirement for line involvement in project planning, leading to line commitment to achieve project goals.
- The need for managerial development in maintenance and low software development activity areas.

This paper presents the solution United Air Lines developed to answer these problems; Two Hat Management, with a first line manager playing two different roles; a line manager controlling manpower resources and responsible for specific functions; and project leader, planning and coordinating the activity of several areas to achieve project goals.

Denver was the location where Two Hat Management was developed and implemented. This facility has a PARS system (programmed airline reservation system) under the ACP (Airline Control Program) operating system on a IBM 360/195. The current application and software systems include more than 500,000 instructions and more than 120 programmer/analysts maintaining and adding to the system by means of projects. Software projects generally exceed six calendar months and either add or modify 50,000 instructions annually. These conditions meet the criteria for large software system projects advocated by Aron.[1]

The paper is organized into four sections to develop and present the material. The first section introduces the environment and scope of the paper. The second section presents the organization structure and functional line management responsibilities. The third section discusses the formal procedures used to implement Two Hat Management in this organization structure. The fourth section summarizes the results of Two Hat Management in the work environment of the last two years.

ORGANIZATION

This section briefly presents the organization structure encompassing Two Hat Management.

Data processing services are provided to many departments and divisions of UAL, Inc., the parent organization of United Air Lines. The divisional organization concept with related profit accountability is the primary method of organization. The Computer and Communication Services Division [CCS] is responsible for almost all data processing activity within UAL. Figure 1 presents the CCS Division organization. The division is headed by a President with a very small supporting staff. There are four line functions reporting to the president: communications and technical services, computer service coordination, resource center and computer services. The latter two include almost all the division personnel and

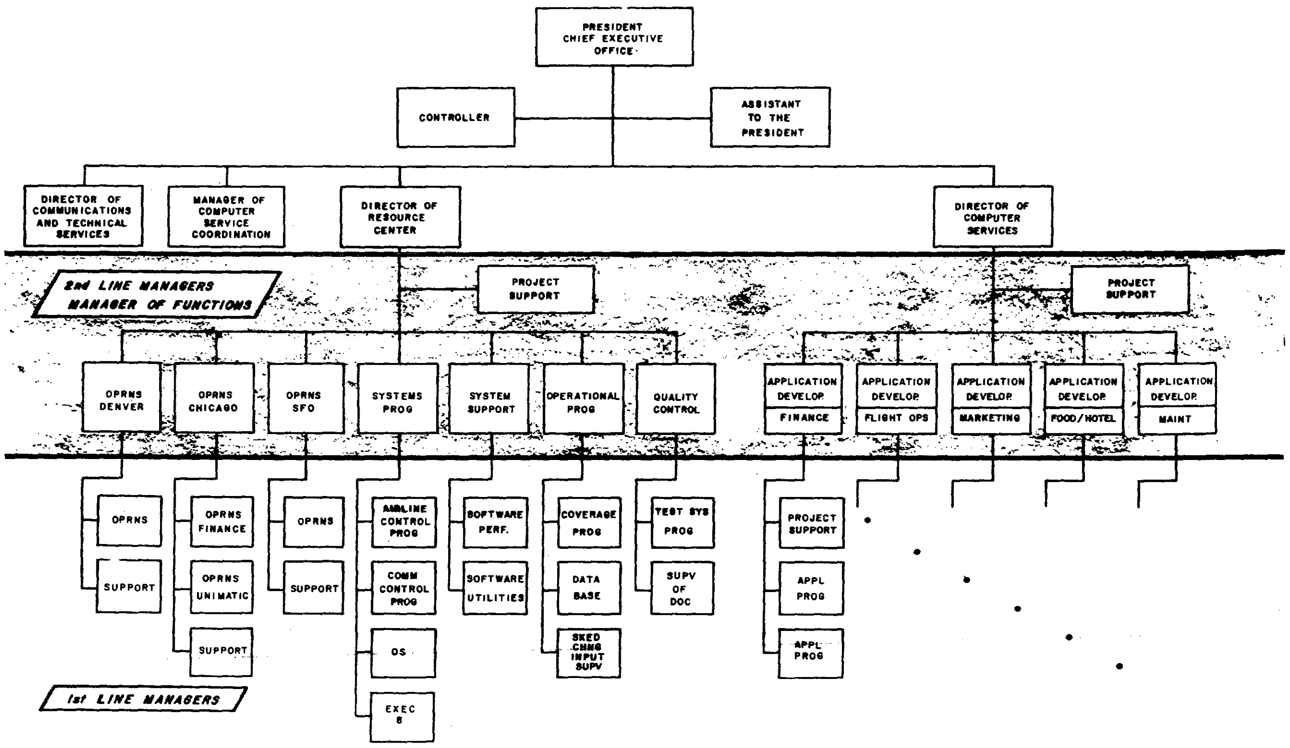

PRESIDENT CHIEF EXECUTIVE OFFICE
CONTROLLER
ASSISTANT TO THE PRESIDENT
DIRECTOR OF COMMUNICATIONS AND TECHNICAL SERVICES
MANAGER OF COMPUTER SERVICE COORDINATION
DIRECTOR OF RESOURCE CENTER
DIRECTOR OF COMPUTER SERVICES
2nd LINE MANAGERS MANAGER OF FUNCTIONS
PROJECT SUPPORT
PROJECT SUPPORT
OPRNS DENVER
OPRNS CHICAGO
OPRNS SFO
SYSTEMS PROG
SYSTEM SUPPORT
OPERATIONAL PROG
QUALITY CONTROL
APPLICATION DEVELOP FINANCE
APPLICATION DEVELOP FLIGHT OPS
APPLICATION DEVELOP MARKETING
APPLICATION DEVELOP FOOD/HOTEL
APPLICATION DEVELOP MAINT
OPRNS
SUPPORT
OPRNS FINANCE
OPRNS UNIMATIC
SUPPORT
OPRNS
SUPPORT
AIRLINE CONTROL PROG
COMM CONTROL PROG
OS
EXEC 8
SOFTWARE PERF.
SOFTWARE UTILITIES
COVERAGE PROG
DATA BASE
SKED CHNG INPUT SUPV
TEST SYS PROG
SUPV OF DOC
PROJECT SUPPORT
APPL PROG
APPL PROG
1st LINE MANAGERS

Figure 1—CCS division organization

resources. Appendix A presents a more detailed description of the organization management and functional responsibilities. The resource center organization is responsible for the computing hardware and system software at four computer sites. Consolidation of resources into a single computer center supported by remote sites is the long-range aim of the resource center, and is reflected in its organization. The computer services organization performs application programming for internal UAL users by means of five application programming centers. A summary of the location, equipment and applications is presented in Table I.

The application centers are in close proximity to the user groups, thus promoting decentralized development, while achieving the benefits of centralized resources, though along somewhat different lines than those described by Tomaszewski.[2]

This organization is characterized by strong first and second line management, accountability for results, with check-and-balance functions incorporated in upper management levels. This is the contrast to check and balance by separation of responsibilities advocated by Smith.[3] The most critical areas are those of interfacing and coordinating. There are three levels of interface and coordination: between first line functions within a department (e.g., first line functions in the resource center), among major departments (e.g., the resource center, the communications and technical services and all computer services' departments), and the CCS division with other UAL divisions (e.g., representing CCS with Flight Operations). This is a responsibility of the Project Support group, which provides project coordination and training services to the line organization. Without strong lines of communication, this particular organizational structure is vulnerable to project discontinuity resulting from lack of coordination. The means of accomplishing this coordination through formal procedures is the major element of this paper.

PROJECT IMPLEMENTATION PROCEDURE

This section presents the procedure used to implement the Two Hat Management concept. It features the meshing of the management organization with checks and balances established by formal procedure.

First line management and work authorization

As emphasized previously, the first line manager has functional responsibility for designated software. Only the first line manager can direct work on the software. The work environment has three types of work; with a reporting system to track progress.

(1) Error Correction—defined by a discrepancy between the expected result (defined by documentation) and the actual result. An FPL (functional problem log) is originated to authorize work.
(2) Modification to existing functions—representing an extension of a current function. A MER (modification

evaluation report) is originated to identify work effort and track implementation, if authorized.

(3) Projects or large modifications—usually a new function to be added to the system. Projects are assigned unique account numbers, and tasks identify work units.

Project management and implementation cycle

The project leader does not control resources or authorize work, he is responsible for managing the project: planning, scheduling, coordinating and reporting progress. Deficiencies and problems are reported to his immediate manager (second line manager), and resolutions are obtained from control group decisions.

The control group is composed of second level and higher department management. The control group reviews project status on a weekly basis, and is responsible for short- and long-term departmental planning and objectives.

The project follows a cycle of definition, planning, implementation and maintenance as illustrated by Figure 2. A Project Workbook is originated and is the repository of all project information.

(1) Project Definition

A project leader is authorized by the control group. Assisted by a project coordinator from the Project Support Group, the project objectives and scope are prepared for control group review and approval. Objectives are the specific goals that the project will achieve, and are used to measure results. The scope is primarily a definition of planning effort required to size the project effort, a plan for producing the project plan. Included in the scope are:

- General statement of project definition.

TABLE I—Resource Center and Application Summary

Location	Equipment	Operating System	Type of System	Applications
Chicago	UNIVAC 1108	EXEC 8	On-line	Flight Operations
	IBM 370/155	IBM OS MVT	On-line, Batch	Finance
Denver	IBM 360/195	IBM ACP*	On-line	Marketing, Food Services, Hotels
	IBM 360/195	IBM OS MVT	On-line Batch	Time-sharing Option (TSO), Remote Job Entry (RJE) for internal UAL users and external customers
San Francisco	IBM 360/65	IBM OS MVT	Batch	Maintenance Operations

* IBM ACP is the Airline Control Program.

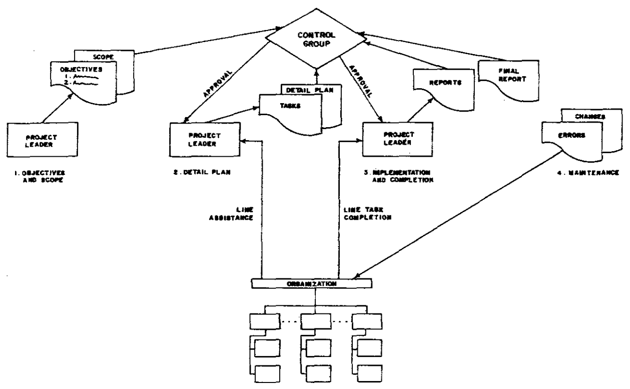


Figure 2—Project cycle

- Investigate requirements to clarify or establish information needed for detail project plan.
- Schedule for producing the detail plan of the project.
- Planning resource requirements, in terms of manpower and computer and related resource requirements.
- Basic approach to performing the project in terms of controls and reporting.

Five types of projects are recognized in this system:

(1) Study project—its objectives are to define the economic justification, relevance and impact of the proposed service on the CCS resources. The end product is a proposal recommending the next course of action to the control group.

(2) Planning project—A planning project is the preliminary step to a very large development and implementation projects. The planning scope of effort and resource requirement is sufficiently large to require a separate project whose products lead to identification and authorization of the main project.

(3) Design project—produces the design development and documentation necessary to support the implementation of a requested service.

(4) Implementation project—includes all work necessary to make the requested service available. Programming, testing, performance measurement, hardware installation: all are valid tasks within this project.

(5) Maintenance projects—a phase which includes maintenance and growth workloads and resource allocations. All workloads of this type are organized into three categories:
  - System maintenance
  - System growth
  - System performance and measurement predictions

It is important to emphasize that planning, design and implementation projects are separated into individual projects based on the control group's estimate of the size, complexity and controls required to perform these phases of the development cycle. When appropriate, these three phases are combined into a single project.

(2) Detail Plan Development

A detailed specification of tasks, resources and schedule of the project is prepared by the project leader for review and approval of the control group. The project leader chairs planning meetings attended by the line managers and/or their technicians. These meetings review the project objectives and scope against the line functions to identify all the tasks and dependencies of the project. Each line area is responsible for identifying and specifying all tasks in their respective areas, and the resources needed to accomplish the task. A functional specification developed by the user is the basis of all planning and resource estimates in the detail plan. In turn, the detail plan provides the developmental and recurring costs which will be charged back to the user.

Plann development is iterative in nature. An overall network is produced first, with subsequent requirements added in. Investigative tasks may be undertaken in order to clearly identify project tasks. The output of the detail plan includes:

- Task definitions—produced by the line area which will have the responsibility for later accomplishment. The line involvement in the planning phase provides the means for strong line confidence and commitment for the project objectives. Appendix B presents the basic task document.
- Dependency network which identifies the sequence of task performance. A PERT chart is another name for this chart.
- Resource requirements for accomplishing tasks. Since these are generated by line areas which will accomplish them, there is no later distrust and disavowal of the estimates.
- A schedule of the task in terms of elapsed time and desired completion date to accomplish the project. The schedule reflects the timing as if there were no other project, with priority conflicts resolved later.
- Major project milestones, which are applied to network and schedule.

Specifically included in the schedule are several critical control points which furnish opportunities for technical review and thorough check-out of the application system product. These include:

- T[illegible]cal Reviews—these are scheduled on an as needed basis. As a minimum there is a review or audit of the detail plan itself. The detail plan is presented to the entire line management for review and comment, including those areas seemingly not impacted by the project. Depending on scope and complexity of the project, additional reviews may be scheduled to review timing, events and dependencies associated with cutover.
- System Test—an integrated test of the coding under regulated and low volume conditions. Scripts which identify inputs and responses are generated by the line area responsible for the code.
- Pre-operational Test—a test of the procedures and actions required by the operations group. This milestone is an acceptance test by operations of the code. Without it, operations will not accept responsibility for production runs. The program documentation is the script, and conditions are created which exercise the documentation.
- Operations Test—a volume test of the code. Its function is to provide the widest possible exposure to the total environment. Again, scripts with inputs and responses are generated by the line area responsible for the code.
- Functional Demonstration—this is an option by the users to conduct a systematic test and check out of the code against the user procedures. The common denominator here is the user-generated functional specification. From it programming develops the programs which is the computer solution of the specification; while the users generate their procedure manual. This insures correspondence between code and user functions.
- Cutover—this is the loading of the code into the system and marks acceptance by the user and completion of the project. Hereafter, all work will be performed on the basis of FPL/MER rather than project.

The detail plan is presented to the control group, along with information relating to the project priority and schedule impact. The control group resolves known conflicts and authorizes the project leader to proceed.

The project leader reconvenes a meeting with the first line managers, identifies the project priority, and requests commitment by the line. The line managers factor the tasks into their work schedules and respond with a completion date for each task. This is the line commitment. If there are differences between the commitment date and desired schedule date which impact the project, the first recourse for resolution is the project priorities. Where this does not resolve the problem, the project leader refers it to the control group who will authorize either a higher task priority, or, accept a later project completion.

These task commitments then become measures of line performance as the project is implemented.

(3) Project Implementation

Project implementation is conducted in an environment where the project leader maintains and coordinates the project. He will report project progress to his immediate manager, but has commitment to the project objectives, irrespective of any line resources he may control. On the other hand, tasks are implemented under line organization control. Programmer/analysts report to their immediate supervisor, and their performance is measured and reviewed in this context. Programmers can work on several tasks and/or other work (MER, FPL).

Higher priority work received by the line manager can impact project work. The line manager reports progress and problems to his immediate manager in terms of task date and manpower commitments. The project leader reports in terms of project orientation and milestones, surfacing problems for control group resolution as required. Independently, the project coordinator furnishes a report of progress to his immediate supervisor, as a check and balance. (A part of the function of the manager of Project Support is that of Devil's Advocate!)

Several major actions occur during this phase:

- Additional tasks may be required for a variety of reasons.
- Extended events may require new priorities—handled, as before, through the control group.
- Modifications to tasks may be required, created by users with the task amendment form (similar to the task description form).
- Reporting of task completion, accumulation of resource expenditures, and accumulation of products as defined in task descriptions.

As testing begins (systems, pre-ops, operations and functional demo), errors are tracked on special FPL reports and fixed on priority basis. Simultaneously, user requests for modifications are evaluated and either deferred until after cutover, or incorporated into the existing schedule and implemented, with schedule adjustments as necessary.

In terms of charge-back, the user will be charged the planned development cost, regardless of actual cost. This, too, becomes a measure of performance, both for the CCS Division, as well as the project leader.

(4) Project Maintenance

The maintenance phase is entered after cutover. Upon reaching this phase, no further work can be charged to the project. All work is either FPL or MER. At this point, the project leader produces a project report, which reviews the project in terms of schedule, resources used and objectives. There is also a critique of the project in terms of items which the project leader feels are important for subsequent projects. The critique can contain suggestions or advice, referencing procedural problems, recognition of work performed or general counsel in the area of leadership.

SUMMARY AND CONCLUSION

With longer-lived computer systems possessing sufficient capacity for the addition of new application software systems, project implementation enters a new phase. Systems continually grow larger, and the organization structure must be able to accommodate both normal system activity (bug fixing and enhancements) as well as development and implementation of new application systems. Implementing new projects in this environment must be done without interposing project organization between the line organization and the ongoing system. The management method described in this paper seeks to incorporate project task activity with the line responsibility for accomplishing the work.

The procedure has been successfully used at the Denver site for more than two years and a dozen completed projects (additional projects are in operation but not completed). The procedure works in the defined environment and has provided four tangible benefits. Management acceptance, enthusiastic endorsement and application of these benefits provide the substantiation for their results.

(1) Better utilization of scarce programmer resources.

Programmers coding in assembly language for a real-time applications environment are scarce. Using the two hat method, first line managers schedule programmer/analysts for a variety of work, including project tasks, without dedicating programmers to projects. In this way, a programmer is busy with FPL (fixing errors) and MER (modifications and enhancements) work while also working on a six-man week project task spread over 12 weeks, for example. The manager will also adjust work schedules dynamically to include new, higher priority work.

(2) Increased technical development of programmers.

Under this system, new people start on FPL work, then working on MERs and later project tasks. In projects, the progression includes modification to existing code and, later, coding of entirely new program segments. Participation in the various test phases provides additional valuable experience. Progression is not sequential; programmers are scheduled for several concurrent work efforts (FPL, MER, tasks) of comparable complexity.

(3) Better software quality control.

The progressive series of tests described in Section 3, provide three to four levels of check-out of actual system quality against the specified requirements. These result in an application system which is stable and user accepted.

(4) Conducive environment for management development.

Twelve out of fourteen first line managers (86 percent) currently are project leaders of significant projects. Of thirteen projects reported in on this paper, there were ten different project leaders. A first line manager always has tasks from other projects to implement, and sooner or later a first line manager will get one (or more) projects of his own to handle. There is no escape from the demanding requirements of project leadership and its attendant interaction with higher management and user levels, as well as with other first line managers. Practical situations abound for management development through experience.

Results

Thirteen projects undertaken between 1970-1973 form the basis of the results reported in this section.

- Work Mix

To put the results in perspective, and to emphasize that the environment is an on-going system, a typical work mix of more than 60 applications programming analysts includes:

Training and non-production time 16 percent
Modifications and error fixing 37 percent
Project development activity 47 percent

Currently, project development activity exceeds 70 percent of planned efforts.

Often the programmers were involved in concurrent types of work (both development and non-development work), and many existing program segments were subject to multiple

TABLE II—Distribution of Project Sizes

Man years of actual work effort	Number of Projects
½-1	5
1-2	3
2-3	2
3-4	1
4-5	1
More than 5	1

concurrent changes. (Modification to one part of the program while fixing an error in another part of the program).

• Project Descriptions

The thirteen projects included two pure study projects. The distribution of project sizes is presented in Table II. Note that the project size is measured by actual man days of work effort, excluding any time spent in overhead activity. In order to put the figures into equivalent people requirements, an increase of 15 percent for overhead activity should be supplied. This additional 15 percent includes categories such as training, sick time, vacation and related items.

Two projects were mainly in the resource center area, seven were in the application center area, and one was a special software development project.

The special project was both the study and implementation of a high level language compiler. During the project, the scope, objectives and resources were increased beyond the original plan. The project was later deferred indefinitely at a point where resource expenditures reached 90 percent of the original estimate and 72 percent of the revised estimate.

In the main, the projects were concerned with implementing new application functions in less than a calendar year, were of moderate size and required tasks from almost all line areas. One significant exception to this was a resource center project converting the system from a 360/65 to a 360/195 main frame, with some significant hardware additions. This project was of such magnitude (requiring more than 17 man years of effort) that it significantly impacted the combined statistical results of all projects. Consequently the results are presented both with and without the impact of this project to avoid distortion and provide comparisons.

• Project Results

The results do not include the special compiler development project which was deferred prior to completion. The two study projects are also excluded because studies (following Parkinson's Law) seem to take all the time allocated for them.

Of the ten implementation projects; eight were on-time (80 percent):

4 were within budget	One was ½ man years, two were almost 1 man year, and one was more than 4 man years of effort.
3 exceeded budget	One was 4 percent more than a two-man year estimate, one took 42 percent more than the 165 man day estimate, and the other took 71 percent more than a 2 man year effort.

An average of 58 lines of source code per man day for 132 days, resulting in 7,800 lines of code, are typical of project results.

Two projects were late, both resource center projects. The major one (main frame replacement) took more than 17 man years versus eight man-year estimate.

Included in the main frame replacement project objectives were:

- replacement of an IBM 360/65 with a 360/195
- installation of a new communication system (front-end processor and software operating system)
- a new communications line control unit (telephone line patch panel)
- modified system software and record type (enlarged control block and modification to a system-sensitive application record)
- conversion of support system hardware (from DOS to OS)
- addition of two new application systems

While the resource overage was more than 200 percent (17 man years of effort versus 8 man years planned), this project was only 3 calendar weeks late. This is a testimonial to the intense involvement by line personnel, resulting from their participation in building the detail plan and commitment to achieving the project objectives. Major problems contributing to the overrun included shortage of skilled manpower, inefficient test vehicles, and lack of coordination and control involving complex, interrelated tasks.

The other late resource center project was a complete reorganization to the on-line files, including increasing the number of disks from 112 to 120. This project took slightly more than twice the estimated 81 mandays. The primary contribution to overrun resulted from extra testing requirements. The impact of test requirements from both projects provide some additional concurrence with the results reported by Boehm.[4]

The main frame replacement project significantly impacted the reported results combining all projects. Table III presents the ratio of actual work efforts experienced on the projects to the planned effort resulting from the detail plan. The figures presented include the results from the resource center, the application center, and the combined result for both organizations. Data is summarized for two conditions: for all projects together, and excluding the results from the main frame replacement project because of its heavy weight.

To interpret these results, note that the resource center actually spent more than twice (2.3 = 230 percent) the

TABLE III—Actual versus Planned Efforts by Center and Organization

	All Projects	Main Frame Project Exclusion
Resource Center	2.3	1.3
Application Center	1.1	1.0
Total Organization	1.6	1.1

resources they had planned, including the results from all projects. Excluding the main frame replacement project, the resource center spent only 30 percent more effort than planned.

The results show that the resource center experienced the greater problems and posed greater potential impact on the system (larger actual experience to planned effort ratios), while the application center work contribution had the dominant effort on overall results (there was more application work effort than resource center).

In conclusion, Two Hat Management works in the specified environment, and the anticipated benefits were achieved. These results were achieved during a period of moderate development work, allowing time for orderly development and fine tuning of the procedures. We are now about to enter a period of heavy software development, which should provide a final test for this management method.

REFERENCES

1. Aron, J. D., *Estimating Resources for Large Programming Systems,* NATO Conference Software Engineering Report Conference, 1969, pp. 68-79.
2. Tomaszewski, L. A., "Decentralized Development," *Datamation* November, 1972, p. 61.
3. Smith, Don, "An Organization for Successful Project Management," *AFIPS Conference Proceedings,* Volume 40, 1972, pp. 129-139.
4. Boehm, Barry W., "Software and Its Impact: A Quantitative Assessment," *Datamation* May, 1973, p. 52.

ACKNOWLEDGMENTS

The author would like to recognize A. C. Edmunds of IBM and L. F. McGrane of UAL, as the fathers of the Two Hat Management concept.

APPENDIX A—UNITED AIR LINES COMPUTER AND COMMUNICATION SERVICES DIVISION ORGANIZATION AND FUNCTIONS

Division organization

The organization chart is as shown in Figure 1. The Computer and Communication Service Division is organized as a profit center with accountability to the parent organization. The internal structure is organized to facilitate the delegation of profit accountability along functional lines.

The division has minimal staff support. There is a controller, responsible for the internal division profit plan and charge-back system for all computer and communications services provided to internal and external customers. The other staff function is a special assistant to the president, providing technical expertise, internal consulting services, and external marketing. There are four line executives: the Director of Resource Centers, the Director of Computer Services, the Director of Communications and Technical Services, and the Manager of Computer Services Coordination who provides liaison between the CCS Division and other UAL organizations: the primary customers of the services.

The resource center is responsible for providing the central site computer resources: the hardware (CPU, memory, files, tape drives) and its operation, the system software (including communication programming), monitoring the use of resources and accumulating data for four computer sites at three locations (Chicago, Denver and San Francisco).

The Director of Computer Services is primarily responsible for providing application programming for internal customer services, by groups called Application Development Centers. Currently there are five Application Development Centers. Two in Chicago, one providing application programming for finance, the other supporting the flight operations functions. Two in Denver, one servicing the airline marketing applications, the other supporting Hotels and Foods Services systems. The last center is in San Francisco and services the aircraft maintenance group.

The Director of Communications and Technical Services has responsibility for the communication lines and interface with AT&T. This includes more than 40 high-speed (2400 baud, or higher) dedicated, leased lines for on-line terminals, an intercompany nationwide telephone network and the installation and servicing of remote site terminals and communications interface equipment.

Division control and policy is set by means of periodic (usually monthly) control meetings with the president and directors, supported and assisted, as necessary by the staff and lower level line management. The purpose of these meetings are for coordination of planning and policy, priority resolutions, review of major project plans and action responsibility for problems. This approach is mirrored in the line organization. Each director holds a weekly control meeting with his staff. The primary difference between the division and department meetings is one of detail. The departments monitor all projects status on a weekly basis, and authorize and review all expenditures of line manpower, with no exceptions.

Resource center organization

The long-range aim of this function is consolidation into a single resource center supported by remote, auxiliary sites as required. The organization structure reflects this approach.

There are second level managers (managers to whom first level managers report) at each of the current resource sites, with second level managers responsible for each remaining function. These are summarized below:

- Manager of Operations—responsible for the central site hardware operations. There are two first line managers, one responsible for the daily operations of the hardware and communications, the other responsible for the support functions of job submissions, scheduling, procedures and operator training.

- Manager of Systems Programming—responsible for all systems and communications programming. There are first line programming managers (managers to whom supervisors, programmer/analysts and technicians report) for each system—one for all OS systems, and for the ACP system, one for the UNIVAC system and one for the Communications Control system in the ACP area.
- Manager of System Support—this responsibility includes all the OS utilities, the ACP utilities, file maintenance, and system performance and measurement (capturing and data reduction of resource utilization), each with its respective first line manager.
- Manager of Operational Programming—the functions of coverage programming (responsibility for keeping the on-line systems up and operating, monitoring and testing new and modified software into the on-line system), data base entry (integrity of data in on-line files) and schedule change data entry (schedule change data for the airline reservations system). Coverage is headed by a first line manager, the other functions are currently led by supervisors.
- Manager of Quality Control—this responsibility includes two functions: the test system and the library. The test system controls and maintains the test tools used in program and system check-out, from initial program unit test in an off-line, single thread environment, up through a volume test of the entire applications and systems. All programs complete strict testing requirements through a series of increasing integrated testing.
- The library provides the control function for tracking the current state of the system. It insures correspondences between program code and documentation. No programs enter the system without authorization from this checkpoint. The library also is responsible for the preparation, editing, classification, recording and filing of system documentation and maintenance of the technical reference library.
- Manager of Project Support—This responsibility is present in both the resource center and the applications development center, but each is responsible to their respective director. By analogy, if the resource center is the bone, the application development center the muscle, then the project support function is the connective tissue tying everything together. Project support provides the interface between and among organization elements, project planning and monitoring service (the IBM Project Management System (PMS) is utilized), project coordination (assisting project leader by preparing project papers, planning, status, meetings and reports) and programmer training.

Application development center organization

This function provides all application programming for UAL user affinity groups. Each application development group is headed by a second level manager and several first line managers. First line managers are responsible for functionally related programs. All errors, modification or new programs are developed under the control of the first line manager. The programmer/analyst is accountable only to the first line manager, including all work performed on projects.

As with the resource center, there is a manager of project support, with similar responsibilities. In addition, there are project support managers reporting to the manager of application development, with the same responsibilities previously described. Here, the project support concept has been extended to a lower level to insure that needed functions and coordination will be accomplished.

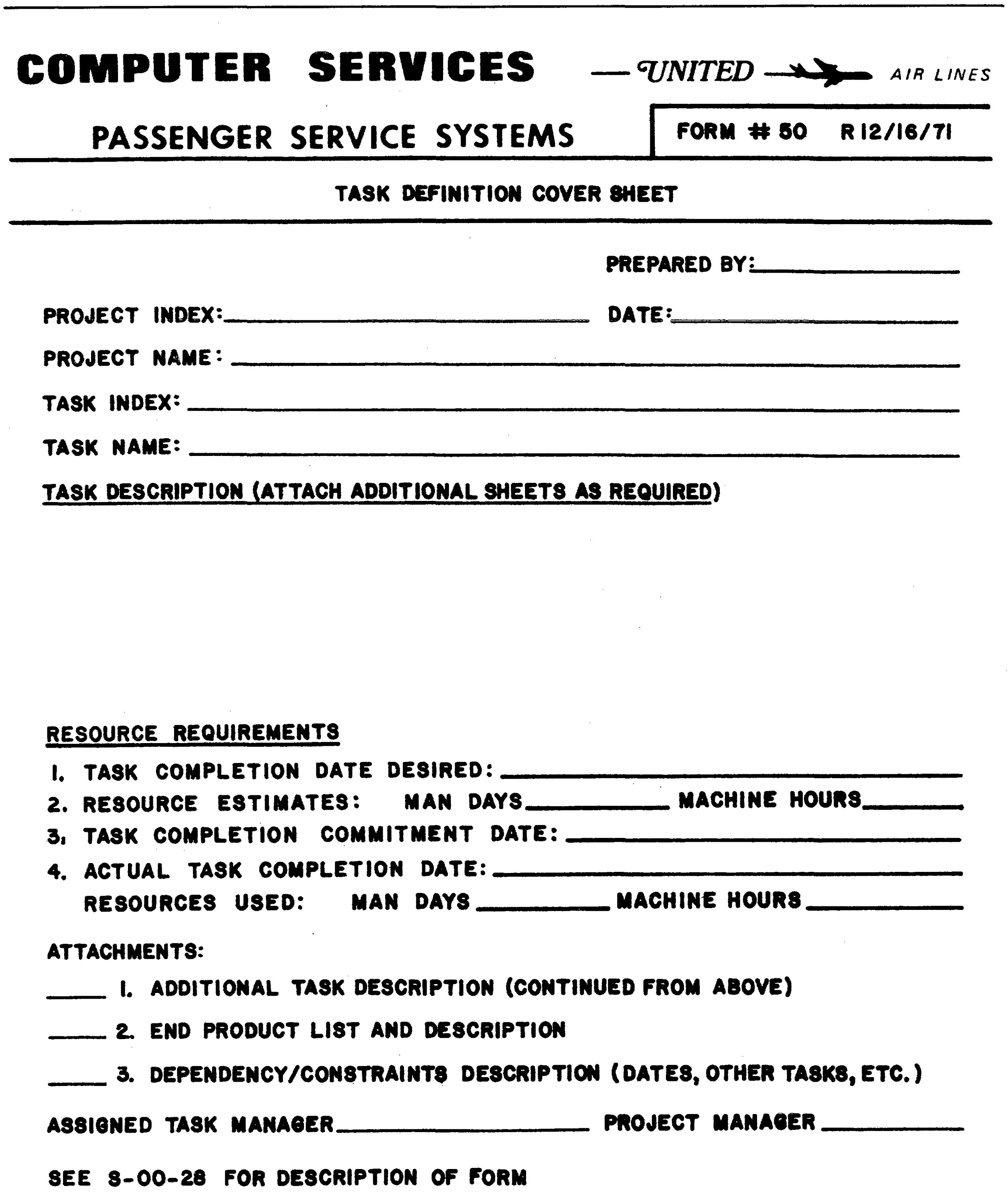
APPENDIX B—PROJECT TASK DEFINITION FORM

This form describes the work to be performed as a part of the project. In order to assure that adequate planning has been accomplished, tasks must be defined in sufficient detail to allow its assignment and to establish its priority within the framework of established workload. The organizational philosophy requires that the task be defined such that it is assignable to a specific first line manager.

The task sheet has four basic parts:

1. Project identification and description—prepared by the project leader, assisted by the project coordinator with input from the first line manager.
2. Resource requirements—completed by the first line manager with his estimates of the workload.
3. Attachments—completed by project leader, used to include supporting material.
4. Task responsibility.

This form exists for the life of the project, first defining the work effort, later recording the actual work performed, and finally, providing a history of completed work and product in the Project Workbook.



COMPUTER SERVICES — UNITED — AIR LINES

PASSENGER SERVICE SYSTEMS | **FORM # 50 R 12/16/71**

TASK DEFINITION COVER SHEET

PREPARED BY: ____________

PROJECT INDEX: ____________ DATE: ____________

PROJECT NAME: ____________

TASK INDEX: ____________

TASK NAME: ____________

TASK DESCRIPTION (ATTACH ADDITIONAL SHEETS AS REQUIRED)

RESOURCE REQUIREMENTS

1. TASK COMPLETION DATE DESIRED: ____________
2. RESOURCE ESTIMATES: MAN DAYS ______ MACHINE HOURS ______
3. TASK COMPLETION COMMITMENT DATE: ____________
4. ACTUAL TASK COMPLETION DATE: ____________
   RESOURCES USED: MAN DAYS ______ MACHINE HOURS ______

ATTACHMENTS:

_____ 1. ADDITIONAL TASK DESCRIPTION (CONTINUED FROM ABOVE)

_____ 2. END PRODUCT LIST AND DESCRIPTION

_____ 3. DEPENDENCY/CONSTRAINTS DESCRIPTION (DATES, OTHER TASKS, ETC.)

ASSIGNED TASK MANAGER ____________ PROJECT MANAGER ____________

SEE S-00-28 FOR DESCRIPTION OF FORM

Controlled testing for computer performance evaluation

by A. C. SHETLER

The Rand Corporation
Santa Monica, California

INTRODUCTION

Because computer systems are still major investments, it is appropriate that effort be directed toward verifying and improving their performance. Previous documents, such as R-549,* recommend an overall framework for addressing computer performance improvements through the use of the scientific method. This paper expands on that base, describing a specific procedure for implementing particular investigations through the technique of controlled testing.**

Though current computer system documentation describes how the different systems function, performance aspects are usually described only in the sales literature and in very general terms. The complexity of these systems make selection, sizing, tuning, and workload characterization a task that is exceedingly—and unnecessarily—risky. Controlled testing can expose information about the execution characteristics of a computer system, permitting the analyst to deal effectively with the assurance that performance prediction and improvement are not random processes.

Controlled computer performance testing is the process of selectively limiting the inputs and operating conditions of a computer system to assist an analyst in discovering and verifying execution characteristics of that system. Controlled testing is necessary because present operating systems contain complex performance-determining variables and relationships that confound the isolation of these relationships when tests are executed in an uncontrolled (normal) environment. The profusion of variables also makes statistical analysis difficult; because their quantity is unknown, unidentified variables can invalidate the conclusions of a testing effort. Controlled testing for computer performance analysis can reduce the variables by explicitly defining the test jobs and the environment in which these tests are executed. By limiting the activities on a computer system, the analyst can isolate and test the performance relationships on that system.

Controlling the environment involves identifying, monitoring, and isolating system activities that are concurrent with test jobs being executed, while ensuring system hardware and software are consistent among tests. The execution characteristics of the computer system being investigated must be identified: hardware, software, interactions, and the restrictions on user and system activity.

Controlling the workload includes specifying the execution characteristics of the test jobs in detail. These jobs are often designed with specific execution characteristics, written for examining specific performance relationships.

WHY CONTROLLED TESTING?

The selection of controlled testing over other performance measurement techniques should be based on the appropriateness of the method to accomplish the purpose of the investigation with the resources available. Because performance analysis usually involves conclusions about millions of dollars worth of equipment and labor, errors can be expensive and the most appropriate cost-effective method should be selected. While controlled testing may not always be the appropriate data collection mechanism, some of the reasons for selecting controlled testing are:

- The reduced side effects of normal system execution.
- The reduced statistical variability of normal system execution.
- The reduced cost, in terms of training, for an analyst.

Controlled testing should be orderly, proceeding within an established framework. Figure 1 depicts the activities of controlled testing.

The activities identified in Figure 1 need expansion for clarification. While many of the activities are relevant to all well designed testing efforts, some are particularly pertinent to testing efforts using controlled testing as the data collection vehicle; the context should reflect this.

ESTABLISH OBJECTIVES/EXPECTED VALUE OF TESTING

The objectives of a controlled test must be appropriate to the scope of the problem(s) being examined and must specify the boundaries of the effort. In addition, some estimate of the value to be received should be documented. The definition of the objectives and expected value is critical; otherwise

* See Bibliography for references.
** This controlled testing procedure is addressed in more detail in R-1436. The concepts described in this paper are elaborated in the Report.

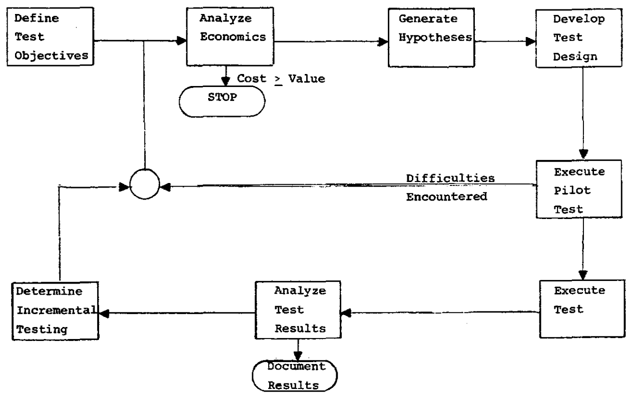


Figure 1—Controlled testing procedure

testing efforts, although interesting, become expensive and futile.

Since the objectives of a controlled testing effort determine the direction for the investigation, they must be established early and with management concurrence. If an investigation has multiple objectives, they should be assigned priorities. Enthusiastic analysts often begin data collection before objectives are clearly defined, only to discover that the data are not appropriate for the investigation.

When defining the objectives, the expected cost and value received must be clearly identified so the cost of the investigation can be balanced against the expected results. Critical resources must be established: the personnel; the machine time; the investigative tools (such as simulators, hardware and software monitors, statistical analysis packages, etc.); and the disruption to operations.

One performance investigation had an objective of establishing the feasibility of removing a core module in terms of the effect of on-line system response time and batch turnaround time. The expected savings in rental for the core module was approximately $90,000/year if the investigation proved feasible in terms of maintaining an acceptable level of user service. Simply removing the module to see what happened was not an acceptable solution to the management (in terms of the lost capacity, user dissatisfaction and the $1,000 reinstallation fee). But the expected value of the investigation was sufficient to allocate a performance analyst, systems programmer, and some blocks of computer time to investigate the problem (costing less than one month's rental of the core module). Once the economic benefits of the situation were established, the resources could be intelligently allocated.

GENERATE HYPOTHESES

Once the objectives of a testing effort have been defined and determined to be economically worthwhile, specific hypotheses should be developed as indicated in Figure 1. These hypotheses will, when tested, become the medium for achieving the objectives.

The complexity of an objective usually determines the extent of hypothesis generation. The important point is not that some specific number of hypotheses be generated, but that the hypotheses that address the objectives be testable. Without testable hypotheses, an effort may simply continue until funding is exhuasted. Without testable hypotheses, analysis will likely be nebulous, and conclusions cannot be supported without a resort to cries for faith in the analyst's integrity.

The analyst should strive for simple hypotheses that are easily tested. Where possible, multiple simple hypotheses should substitute for a single complex one. The relationship between each hypothesis and the objective should be stated and documented, even when it may appear obvious. Otherwise the hypotheses may be investigated that are irrelevant to the objective. Explicit assumptions should be examined to be sure they are not, in reality, hypotheses for testing.

The following information is necessary for documenting hypotheses:

- Hypothesis: What the analyst proposes to test.
- Relationship: The relationship between what is being tested and the objectives of the performance evaluation.
- Assumptions: The explicit and implicit assumptions about the hypothesis being tested.
- Analysis: The information (data) and processing to validate the hypothesis.
- Data: Description of the collection mechanism.
- Alternate Hypothesis: What the analyst believes is true if the hypothesis is rejected.

TEST DESIGN

The types of analysis to be employed in test evaluation must be defined and documented along with a detailed description of data required for validating hypotheses. Then the analyst can define the tests for gathering the data. The analysis should include tests for an invalid hypothesis; without considering this, a series of tests could be executed which do not generate conclusive evidence that a hypothesis is either valid or invalid.

Generating a test design calls for ingenuity if testing is to be performed efficiently. Properly designed tests may overlap and collect data for evaluating several hypotheses. The environment—both system and workload—must be specified during the test design. The details of both hardware and software must be documented before the analyst moves on to specifying assumptions, variables, and a detailed operating procedure.

A configuration diagram of the hardware most clearly identifies the system hardware. The devices not required for testing should be shaded to aid visual identification of the test equipment. Obviously, the operating system and its version must be identified. In addition, descriptions of installation variations from the unmodified version of the operating system are appropriate. These may include:

- A description of resident vs non-resident portions of the monitor.

- A description of installation modifications to the standard system, and how these may affect the testing.
- A list of subsystems, special security monitors, bulk media conversion routines, etc., that will affect the testing.
- The dimensions of resident systems (such as time-sharing) that will be active during the test.

If possible, a mechanical description of the system state at initialization of the testing should be included (i.e., an operator's log display that lists the supported and existing devices).

All assumptions relating to the testing effort should be listed explicitly. This list will aid in evaluating the testing procedure. Assumptions might be identified that relate to the hypotheses, the analyses, the hardware, the software, the tests, or the testing environment.

The analyst should explicitly identify the variables that are expected to change during execution of the testing procedure. The relative significance of each variable may not be easily determined and many of them may seem trivial. However, they must be spelled out because a presumably trivial variable can, in particular situations, have a nontrivial impact on testing results or on subsequent analysis. Without initially listing the possible variables, the analyst could waste time during the test execution and analysis detecting these variables after side effects have been encountered.

Develop detailed operational procedure

An operational procedure sets forth in a document the planned activities that operators and analysts will execute. This "cook book" for executing the tests is included as part of the test design. Without such a document, a test can deviate in subtle but significant ways from the analyst's intended course. Time is always limited during test execution and haste can result in skipping a critical step in system setup or test execution if the operational procedure is not carefully specified.

Before a test, the order of operations can be reviewed to ensure the procedure. Where appropriate, the operational procedure can be modified to reflect corrected assumptions. The detailed operational procedure can take the form of a checklist to be used during test execution to verify the planned activities. Using this detailed procedure reduces the time used to mentally verify that generally stated conditions have been met.

Variability

When designing tests, the analyst should assume that statistics will usually vary in test results. The variability is currently unavoidable, but is not an insuperable handicap. Variability can be identified and quantified in the following way:

- Execute the tests several times to identify the extent to which standard system variability affects the data collected by the test set.
- Examine the test environment for variability factors (i.e., scheduling, spooling, resource allocation). Once such factors are detected, the test procedure can be examined for ways to reduce variability.

PILOT TESTING

As Figure 1 indicates, once the analyst completes his test design, he should conduct a pilot test. This permits a validation of the design before he attempts an actual test. The objective of pilot testing is to discover the problems with the procedure; the analyst is seldom disappointed by finding the procedure perfect.

If possible, the analyst should do all debugging prior to stand-alone pilot test time. Programs, control cards, and input parameters should not be debugged during stand-alone time; these tasks are not time-critical and can usually be performed more easily in the working environment. All jobs should be completely tested, ready for use prior to pilot testing.

The analyst should verify that the system state (environment) is constant. This is particularly important when the test will be executed in more than one session; without it, unexpected conditions may add new variables and confound any conclusions. This constant system base should be established prior to actual test execution (that is, during pilot testing). The operational procedure should include steps to verify the base state.

Redesign experiment

Pilot testing of the experiments usually results in redesign of some of those experiments or the procedures. Because several pilot test sessions may be necessary, all output should be kept until the experiment redesign is complete. In this way, all information is available for reviewing the revised design. As indicated in Figure 1, pilot testing should be repeated as often as necessary to refine the total experiment.

Minor modification of the assumptions stated in the initial test design are occasionally a result of pilot testing, but the analyst should expect the revisions to the detailed procedures to be extensive. After executing a pilot test, the analyst should document the revised detailed functional and operational procedures. Included in this documentation should be modifications to the assumptions (based now on experience with the environment) and an identification of the critical variables with a projection of their expected impact.

TEST EXECUTION

When an analyst completes the pilot tests and arrives at the final design, a reasonably precise description of this test procedure is already in hand. To ensure that the operational procedure is closely followed, the analyst must be present during the test execution as an active participant rather than an observer. Ideally, the analyst performs all operations—mounting tapes, operating the console, feeding cards, etc. An inability to perform these functions may indicate a critical lack of understanding about the system. (Experience indicates that a few hours with a senior operator is usually sufficient to acquaint an analyst with the essentials of operating any system for controlled testing.) For example, lack of familiarity with console commands might indicate the analyst is unaware of which system parameters are subject to modification and are important to performance determination.

Some analysis of performance data must parallel the test execution. This allows the detection of "unexpected" events. In addition, the test results can be monitored for indications that the tests are proceeding as expected.

Time should be reserved for reruns and correcting errors in procedures. Problems are unavoidable in even the most carefully prepared efforts and the analyst must plan for them. The amount of time for rerun and error correction should be increased as (1) the elapsed time between pilot testing and actual test execution increases, and (2) the number of uncontrolled variables increases. For example, using one system to prepare for the test effort and another to execute the final tests will dramatically increase the probability of errors (e.g., where all the preparation is done on a system at the user's installation, and another system—a new one—in a vendor machine room is used to run a test stream).

ANALYZE TEST RESULTS

The data produced during testing should be analyzed on an informal level as the test is being executed. Thorough analysis of the test results should begin, if possible, within 24 hours of the test period. This leaves the testing experience clear in the mind of the analyst; although notes are generated during the experiment, the details will fade with time. The test results should be examined with particular emphasis on the following:

- A desired level of control was specified in the testing procedure; the test results should indicate the level of control attained and the status of the critical variables.
- The system components supposedly being held constant should be examined to determine whether they have been changed.
- The system components being compared, tested, or both, should be examined to determine the test's applicability.

The validity of the original hypotheses can be determined only by a careful analysis of the actual test results. This analysis may be extensive and part of it may take place long after the test. However, each hypothesis should be examined in some detail immediately, because a long analysis may be required for one or more of the other hypotheses. More thorough analysis can be done after the immediate analysis where the special conditions requiring further work are identified.

The results of the analysis of the test data should be documented in terms of the hypotheses being tested. The implications of these results to the objectives of the performance investigation should also be documented. As indicated in Figure 1, results may be incomplete because some hypotheses are invalid or the test execution revealed additional questions and additional analysis is required to determine the additional testing to complete the investigation.

DETERMINE INCREMENTAL TESTING

Analysts should expect some hypotheses to be untrue; when this is so, it may be appropriate to generate new hypotheses. In addition, the test procedure may turn out to have been inadequate (regardless of the most exhaustive preparation). The test procedure, or experiments, may require redesign, but the analyst should only attempt to redesign the tests if the expected value exceeds the expected cost with respect to the original—or a revised—objective.

An evaluation of new hypotheses must include the collection of new data. However, a hypothesis generated to match a particular set of data can be proved only with new data.* Otherwise, the "hypotheses" are merely descriptions of existing data.

NOTE

Figure 1 describes the procedure for implementing controlled testing, there are two related special topics that require expansion, Testing Processing Environment and Test Stimulation. These topics are addressed specifically in Appendix A and B, respectively. The explanations in these appendices should add clarity to the test design phase for implementing the controlled testing procedure.

BIBLIOGRAPHY

Bell, T. E., B. W. Boehm, and R. A. Watson, *Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort*, The Rand Corp., R-549-1-PR, November, 1972.

Bell, T. E., *Computer Measurement and Evaluation—Artistry, or Science?* The Rand Corp., P-4888, August, 1972.

Bell, T. E., *Computer Performance Analysis: Measurement Objectives and Tools*, The Rand Corp., R-584-NASA/PR, February, 1971.

Boehm, B. W., *Computer System Analysis Methodology: Studies in Measuring, Evaluating, and Simulating Computer Systems*, The Rand Corp., R-520-NASA, September, 1970.

Bookman, P. G., B. A. Brotman, and K. L. Schmitt, "Measurement Engineering Tunes Systems," *Computer Decisions*, Vol. 4, No. 14, April, 1972, pp. 28-30.

Kolence, K. W., "A Software View of Measurement Tools," *Datamation*, Vol. 17, No. 1, January 1, 1971, pp. 32-38.

Lockett, J. A., A. R. White, *Controlled Tests for Performance Evaluation*, The Rand Corp., P-5028, June 1973.

* If a new hypothesis is generated from examining existing data, the analyst may be able to reject it based on an analysis of these data, but he cannot accept it.

Mayo, E., *The Human Problems of an Industrialized Civilization*, Macmillan, New York, 1933.
Roethlisberger, F. J., and W. J. Dickson, *Management and the Worker*, Cambridge, Mass., Harvard University Press, 1939.
Seven, M. J., B. W. Boehm, and R. A. Watson, *A Study of User Behavior in Problem Solving with an Interactive Computer*, The Rand Corp., R-513-NASA, April, 1971.
Sharpe, William F., *Economics of Computing*, Columbia University Press, 1969.
Shetler, A. C. and T. E. Bell, *Computer Performance Analysis: Controlled Testing*, The Rand Corp., R-1436-DCA, 1974.
Shetler, A. C., *Human Factors in Computer Performance Analysis*, The Rand Corporation, P-5128, 1974.
Warner, D. W., "Monitoring: A Key to Cost Efficiency," *Datamation*, Vol. 17, No. 1, January 1, 1971, pp. 40-42ff.
Watson, R. A., *Computer Performance Analysis: Applications of Accounting Data*, The Rand Corp., R-573-PR, May, 1971.
Watson, R. A., *The Use of Computer System Accounting Data to Measure the Effects of a System Modification*, The Rand Corp., P-4536-1, March, 1971.

APPENDIX A—TESTING PROCESSING ENVIRONMENT

While controlled testing can reduce the number of variables that confuse analysis, total control divorces an experiment from the workload characteristics of the real job stream. Experiments can be designed to combine tests in a totally controlled environment with tests in a partially controlled environment to assure relevance of the results to the current workload on a system, when this relevance is a requirement.

Performance investigations that combine partially and totally controlled testing environments can provide assurance of the relevance of test results for objectives that are difficult to prove in other ways. An effective procedure for testing an objective requiring applicability to a specific system has been to apply two varieties of specific hypotheses: those postulating abstract relationships between hardware and software, and those postulating the effects of these relationships on processing the usual workload. The results of testing the former under a controlled environment can direct testing the latter hypotheses under the system environment to ensure applicability of the results to the current environment.

Tests with both partial and total control are pursued in the same basic framework, using the same techniques, and observing the same caveats. However, some special considerations apply to tests in each environment.

Partially controlled workload environment

When the analyst must use the normal workload environment for an experiment, unpredictable elements must be addressed including the operators, users, and shifting workload. If these are ignored, the results may be more indicative of unpredictable human reactions than of the effects being investigated.

If the users or operators are aware of experiments in progress, they may respond by changing their behavior and create an uncharacteristic environment (often referred to as the Hawthorne effect). Spurious workload changes can also invalidate experimental results. If a testing period coincides with a particular event, such as annual accounting and reports, the results can be deceiving when applied to the standard working environment. Normal fluctuations in workload over short periods (one or two days) can create the same effect. The analyst must remember that a computer system workload seldom reaches a "steady state." The human behavior and workload characteristics of a system during a short test period must be compared with characteristics over a longer period. A formal comparison period with appropriate controls should be used to reduce the probability of invalid results caused by autonomous changes.

Totally controlled environment

Specifying and generating the desired environment is critical in an experiment with total control. If the objective of an experiment is to examine specific system interactions, a relationship between the artificial and real workload need not be established.

A cooperative effort between performance analysts and system maintenance personnel is more than desirable; it can be imperative for controlled tests to be productive. When a hypothesis involves system-related interaction, implicit and explicit assumptions and conclusions that are consistent with the system's operational characteristics can be assured. However, the analyst must view all untested statements about performance characteristics with skepticism; systems are subjected to continual change, so opinions about performance characteristics may be no more than strongly-stated computer folklore.

Selection of a "background" workload may be necessary for some controlled workload experiments. A background workload consists of jobs designed to create some degree of system activity. The activity created by these jobs is important to the controlled test, but the specific job characteristics are considered unimportant. Background alternatives include:

- No background. Only the tests are executed.
- Background spooling.
- Activating an on-line system through artificial stimulation.
- Activating background jobs with specified execution characteristics.

Detailed considerations

After selecting the general approach for the degree of workload control, the analyst must choose the details of the processing environment. The following must be included in the considerations for both partial and total control:

- The hardware and software configuration.
- The system state (including the activity level of system

functions and sub-systems such as spooling concurrent with test execution).
- File placement on peripheral devices.

In addition, the availability of computer time, the work schedule, the purpose of the experiment, and the quality of the data being collected to validate hypotheses will influence the environment selection.

APPENDIX B—TEST STIMULATION

Test stimulation involves loading a computer by imposing programmed activity that has specific resource utilization characteristics. When an analyst is performing controlled testing, the loading stimulates well-defined system functions in specific ways. The total spectrum of test stimulation includes not only conventional batch benchmark jobstreams that have been used for equipment procurement, but scripts used for stimulating on-line systems.

BATCH STIMULATION

Batch stimulation can consist of (1) a job stream selected from an actual installation workload to represent that workload (usually called benchmarks); (2) a job stream selected to generate a sample of work, not intended to represent the workload realistically; or (3) a synthetic job stream that generates desired execution characteristics but does not attempt to replicate an installation workload. Each type of job stream has a different, but valid, purpose.

Representative job stream (benchmark)

When a subset of the normal workload (with the same average component utilizations) is desired, the accounting data generated by the benchmark should be examined to determine that these data are approximately equivalent to the normal job stream accounting data. Once the objectives, hypotheses, and procedures for an experiment are clearly defined, many of the accounting statistics may be irrelevant and can be ignored, or additional statistics may be added. However, reports resulting from an investigation using a benchmark-type job stream must include adequate caveats about not extending the results beyond the workload of the system being tested although they may have applicability to other systems processing similar workloads.

Specialized job stream

Specialized job streams are often used to investigate unusual phenomena that are observed. A job stream selected to generate a sample of the workload does not permit the same analysis and the results cannot apply specifically to any computer system workload. A nonproportional job stream should not be an excuse for analysts to do sloppy work; valuable results can be produced if these job streams are correctly used. A specialized job stream is used in establishing system behavior patterns when a nonrepresentative, abnormal subset of jobs are run (for analysis of very specialized relationships). For example, an investigation to determine the effectiveness of an internal operating system modification required a specialized job stream to validate the implementation of the modification.

Synthetic job streams

Synthetic jobs are designed to stimulate the system to exercise specific components in well defined ways. Some advantages of using a synthetic job stream are:

- The synthetic job can be designed with execution characteristics that are totally specified and understood.
- Internal data collection and data reduction can be designed into a synthetic job; it can serve as both the stimulator and the monitor.
- Compatability between machines and vendors can be insured for the bulk of the job, and the analyst can perform comparison studies.

ON-LINE STIMULATION

Generating test stimulation for on-line systems is difficult, and the development of procedures for this activity has been minimal. One of the primary causes for this situation is lack of information about normal input (command sequences from on-line sessions with associated "think-time" gaps) to facilitate duplication for on-line stimulation. This lack means that on-line test stimuli are usually generated by scripts specifically designed to test whether certain functional characteristics are present, rather than to test a richer variety of performance related activities of on-line systems. Such script are usually created without even the crudest measured data and stored on punched paper tape or printed for manual transcription during experimentation. They are costly in terms of the time to produce and execute.

In addition to these crude techniques for script input, sometimes a system is tested with another computer system as a stimulator. The stimulator generates input from one or more scripts and inputs this to the computer system under test, replicating multiple terminal activity. The initial appeal of such schemes is placed into perspective once the design, implementation, and application costs have been determined.

Script consideration

Crude techniques should be used initially in an investigation. The analyst can usually devise a minimum set of on-line commands, manually use them, and then expand the simple scripts as experience dictates. For a context-dependent system, the analyst should initially determine the minimum number of steps that can be meaningful, based on the independence of the script's performance from potential commands before and after it. In context-independent portions of a system, single-line scripts are desirable because they make analysis easy.

Some simple data collection for existing systems can be implemented without resorting to sophisticated tests. Since users of on-line systems are accustomed to people in the immediate vicinity of the terminals, they can be observed without disrupting their activities or altering the situation being observed. Some of the items to observe are:

- The incidence of using each subsystem (command mode),
- The relative incidence of each command,
- The rate of command submission,
- The rate of typing,
- The evenness of input (whether activity occurs in bursts or is relatively constant through time), and
- The incidence of active but idle terminals.

When better information is needed, automatic data collection facilities are often available for aggregate information to reveal loading. Combining the data from an automatic collection facility and personal observation of users may be necessary when a data collection facility is not available to obtain more detailed data.

A very basic rule has proved important in on-line stimulation: always begin with a very simple stimulation, even when a complex type is clearly required. The initial stimulation must be simple enough that the analyst can explain each response of the system to it. On-line systems are complex and understanding them is difficult; the danger of complex scripts is that they lead to incorrect conclusions.

Installing an on-line information system in a manufacturing environment

by THOMAS J. ARCHBOLD

International Harvester Company
Hinsdale, Illinois

The Construction Equipment Division of International Harvester Company in 1971 developed a broad program to dramatically improve the material scheduling and shop floor control functions at the Melrose Park plant. The basic objectives were to improve efficiency and to be more responsive to customer demands for our construction equipment. The system that existed then provided manufacturing with data that could be from several days to several weeks old. An up-to-the-minute current status of material availability within the plant was not available. Discrepancies and error conditions of all types were extremely difficult to resolve because of this time delay. Changes to our manufacturing schedules to meet sales requirements were further complicated by the lack of current information.

The program that was designed to resolve these and many other manufacturing problems is called MICS, Manufacturing Information and Control Systems.

The primary objective of MICS is to generate realistic and economic manufacturing schedules for the machining, fabrication and assembly departments. In order to accomplish this objective, accurate and timely data must be available on the status of each part and assembly as they move through the 120,000 manufacturing operations to final assembly. This requirement dictated a system that would collect, analyze and record quickly changes in the status of material as it moves through the plant. The overall stock status of all parts and assemblies would have to be accessible to manufacturing personnel on request. The system must minimize the clerical work performed by plant supervisory personnel and maximize the manufacturing information available to them as required.

The major portions of this comprehensive system are in operation at the Melrose Park, Illinois, plant of the Construction Equipment Division. There are more than 100 data collection devices, typewriters and other terminals online or interacting with a remote computer 20 hours a day, six days a week. The remote computer is located at International Harvester's Corporate Computer Center, Hinsdale, Illinois, approximately 10 miles away.

A brief overview of the manufacturing facilities and the products manufactured will provide an insight into the complexity of this manufacturing requirement. The facility is located on a 125 acre site in Melrose Park, Illinois. There are 2.4 million square feet of manufacturing floor space. The plant has 70 interdependent departments, 2,200 production machine tools, approximately 4,000 employees, 40,000 purchased and manufactured parts and 120,000 manufacturing operations. A complete line of heavy construction crawler tractors and diesel engines are manufactured and distributed worldwide. Each end product has a wide variety of customizing internal and external attachments. The value and complexity of the product coupled with the wide range of significant attachments available on each product, add additional dimensions to the overall material management and manufacturing requirements. Manufacturing must be flexible and responsive to changes in production schedules as may be predicated on approval of major highway or other construction projects. Our ability to deliver the specific customized product required on a specific date, in many cases determines whether or not IHC obtains the order. Our recent order with the Soviet Union for our TD25C Crawler Tractor required special Arctic insulating and lubricating equipment so that these tractors would work at temperatures from a minus 60° to over 100°.

Before MICS, the computerized material management system that had been systematically developed since the late 1950's had been relatively effective and in many aspects a vanguard in many of the material control disciplines. However, the total system was oriented to either weekly or monthly shop and end product schedules.

Why develop a new system? In 1968, divisional management expressed a strong desire to improve the accuracy of manufacturing data and also the responsiveness of manufacturing to react to relatively short notice market needs. Extensive investigation by a five-man functional task group recommended a pilot MICS program be installed in a large complex machining department at Melrose Park. This "laboratory" was designed to evaluate many aspects of shop floor control and overall material management concepts. The pilot system which monitors 120 work stations has been successfully in operation for over two years. This system provides instant communication between the worker, foreman and a central control room. It mechanically collects production counts for the material control and incentive payroll systems. Machine downtime, setups, teardowns and other allowance conditions are accurately monitored by the computer. The integration of this pilot machine monitoring program into the overall MICS program will be discussed later.

The positive results gained in the machine monitoring pilot program and the continuing requirement for more accurate data throughout the plant resulted in the development of a plantwide total material management program.

What are the objectives of MICS? The primary objective of MICS is to develop economical manufacturing schedules for the 70 interdependent machining, fabricating and assembling departments. Effective scheduling in a dynamic environment required more timely information on the location and availability of all parts, assemblies and raw material. The investment in MICS was made to do the following:

1. Parts tracking is accomplished by accurately monitoring the availability of the 40,000 manufacturing and purchased parts in the production stores and manufacturing departments. Total parts availability is determined by exploding daily the produced crawler tractors and engines and reducing the availability of these assemblies for further manufacturing. The low level assemblies are exploded into their simple parts as each worker reports his production through data collection units.
2. Job order loading on individual machine tools or machining centers will be scheduled on the basis of setup costs and compared to carrying costs. Job order loading will provide the vehicle where EMQ can be effectively introduced onto the manufacturing floor. Job order loading of machine centers will also improve the predictable parts availability to meet production schedules.
3. The availability of assemblies and/or the simple parts will be predetermined at the time the order is received for that assembly. MICS will determine shortages by each customer's specific build ticket. Expediting and parts chasing will be reduced considerably.
4. Mechanized requisition of parts and raw material from production stores or the machining departments is now possible. The specific start of each machine operation is now predictable.
5. Based on the works approved end product forecast, attachments are currently being mechanically forecasted. This program is a major system in itself and will not be discussed in this paper. It is an integral part of the total MICS system.
6. The capacity planning phase of MICS will convert the end product forecast into machine tool and manpower loads for each department by weeks for a 13-month period.
7. The order entry system will generate a customer order and build ticket in machine readable form. The order, with its attachments, will be compared to the forecasted schedule for finished machines and attachments, and the most economical production slot in relationship with the customer's order date will be established. Reslotting of customer orders will be mechanically accomplished as schedules are changed.

How was this feasibility developed, presented and finally approved by top management? The above objectives were specifically delineated by the task group to executive management. A preliminary feasibility detailing the overall concept of MICS with estimated costs and savings was presented to executive management. The acceptance of MICS as a viable project was conditioned on specific resources being spent in areas that may be either overlooked or understated in feasibility proposals. These conditions were:

1. That a consulting training firm be called upon to develop and implement a training program.
2. That 20 stock status specialists be hired to analyze and correct production discrepancy conditions immediately as they occurred.
3. That a MICS Project Manager be assigned full time and report directly to the Divisional Vice President of Manufacturing.

In depth, strategic planning on all phases of this enormous project began with executive approval in November of 1971. Current manufacturing techniques played an important part in developing the implementation program. Because material moves through Melrose Park plant on a continuous flow basis, each department functions somewhat independently. There is a complex interaction on material movement, material requisitioning and material scheduling. This environment necessitated that each aspect or phase of MICS be implemented over the entire plant in as short a period of time as possible. Functions that could be separated were split into three phases. This basic approach allowed us to monitor the reliability and the results of each phase as well as receive the benefits of each phase, and because of the interrelationship of the various phases, any changes that were required during the parallel or implementation time period could be more economically reprogrammed.

What are the MICS phases? MICS Phase I provides the data base and data communications capability to enable management to maintain complete control over all purchased and manufactured parts, from the basic castings and forgings through lower level assemblies to the end products. Phase II extends MICS capabilities to enable the system to slot and reslot customer orders, to establish uniform assembly line sequences, and to predetermine the availability of parts for scheduled assemblies. Phase III further refines the system to calculate economic manufacturing lots at all levels, schedule production, sequence jobs, requisition the necessary parts and components, and recommend piece part production start and stop dates.

The MICS project spans a two-year period. Each phase was designed to overlap certain activities in order that maximum utilization of manufacturing and systems personnel was attained. Although the overall concept was approved, a major planning effort remained. All levels of manufacturing personnel had to be motivated to change—to accept this new technique of manufacturing control. We developed, with the training consultant, Wilding Incorporated, Division of Bell and Howell, a 30-minute video film with key manufacturing management as the actors. The reasons for MICS, what MICS would do, and the functions of various computer equipment were addressed in this film. The film was directed to the foremen or first line manufacturing management. After viewing the film, there were buzz sessions with small groups of foremen conducted by industrial psychologists. Some of the recommendations from these sessions were incorporated into MICS. Training and systems manuals were prepared. Techniques were developed to train the thousands of plant personnel who would be using the system.

Now that the forward momentum had been established, detailed activities and implementation schedules would have to be developed to support each phase. Such questions as—What type of inquiries?—What are the error condition parameters?—What specific audit trails are required?—What aspects of the total system require backup or redundancy? This definition would then be converted to programming estimates, training requirements, and equipment installation dates and all other related tasks. Critical resources such as systems expertise could then move from one phase to another when they completed their portion of an earlier phase.

Each major area, such as systems development, programming, training and equipment installation, were divided into sub-tasks which were manually perted. Progress in all areas was reviewed weekly, and management was apprised of the progress through an exception report.

Throughout all phases of MICS, special computer analyses were designed and programmed to evaluate various aspects of the manufacturing data base. Simulations were used to determine online response and offline processing times. The "one-time" effort in analyzing data and alternate processing techniques is more prevalent in all our systems design today. This kind of analysis is almost essential in the development of online systems because of their hardware/software complexity and generally high cost. Special programs were developed to assist the user in his conversion from the old to the new system. For example, data from the new and the old were displayed on the same report for ease of comparison.

Because Melrose Park plant operates under continuous material flow concept and each work area is depending on others for materials movement, requisition, scheduling, etc., the implementation of MICS was difficult.

In order to maintain the integrity of the manufacturing data, implementation of each phase would have to be accomplished in the shortest period of time. Any installation delays would create an excessive hardship on the Stock Status Investigators in resolving any discrepancies between the MICS information and the current system at that time.

The primary objective of Phase I is to collect and maintain availability information on all purchased and manufactured simple parts and assemblies. This data includes all castings, forgings, purchased finished simple parts and assemblies, and manufactured assemblies.

During Phase I, data collection procedures were introduced. During this department-by-department installation, the prepunched job assignment card system would be running in parallel. Duplicate employee earnings and production analysis statements were prepared and analyzed to insure accuracy of the new system. In order to maintain good employee morale, this duplicate reporting time period had to be as short as possible. Prior to the actual department parallel, an extensive training effort was made to assist the foremen and expediters in that department. A flexible front end program was used to move a department from the old system to a combination of the new and the old processing and finally to the new program. This flexibility was maintained so that a department could be put back on the old system if unforeseen difficulties arose.

Each machine tool schedule is keyed to the component parts required to fulfill its schedule. Component availability quantities and locations on any assembly or part is available on inquiry on 32 MICS typewriters located throughout the plant. In order to maintain reliable data, all simple parts and assemblies must be carefully tracked through their various phases of processing. This further reinforces the requirement to install the data collection and the typewriter/punch units as quickly as possible.

As a simple part or assembly is completed and then reported by the employee through the data collection station, the next department is determined from the routing file data base and an authority to receive card with the quantity made is punched out in this department. When the material is received, it is verified against the quantity on the card and the card is inserted into the data collection device to complete that portion of its routing through the shop. For example, the reporting cycle for a rough water pump impeller begins when it is received in the castings yard. The system shows the impeller as being available for any part that requires such a casting. A requisition to the casting yard for the impeller and notification of its receipt in the requisitioning department changes the impeller's location on the central data base. However, after the initial production operation is performed by the first production department, the raw casting will be no longer available in the system for other part numbers that might use this stock. It will be available only for subsequent operation of this particular impeller part number.

When the final production operation is completed, the impeller will be shown as available to the next higher level of a particular water pump assembly. When the impeller is assembled into the water pump assembly, it loses its identity and availability as an impeller part number. The system will continue to track the impeller or water pump from department to department, showing the availability of the total water pump. All other parts and materials used in this finished product are tracked in a similar fashion. The level-by-level component tracking provides answers at any time to inquiries on the availability and location of raw castings, forgings, purchased finished parts and assemblies, and manufactured assemblies.

A foreman, expediter or plant manager can use the 32 typewriter inquiry terminals to ask such questions as:

What is the availability of the finished components for a given part number?
What is the finished inventory of a given part number?
What is the in-process inventory?
What is the date and quantity of the last made figures?
What is the production schedule for a given part number?
What is the opening inventory or accumulated scrap of a given part number and operation?
When and how many adjustments have been made to the given part number and operation?

These and a variety of other questions concerning various fixed information about a part number such as its routing and process time are available on request 20 hours a day, six days a week.

When MICS indicates that stock is available, it reflects all actual usages. Simple parts that are already used in higher assemblies are shown as unavailable. Raw stock availability is reduced as the raw castings are requisitioned from the storage yard. Availability of parts is maintained by a series of unloading stages. Simple parts disappear as they are assembled into a higher level sub-assembly by exploding the Bill of Material the moment the employee reports the made quantity in the data collection unit. This unloading technique is also used when parts or assemblies are moved.

Parts and components that are used on the assembly line are unloaded by exploding the finished engines and tractors daily.

The processing points that require unloading and reducing the availability of material are:

1. In the warehouse when purchased or rough castings are moved.
2. In a machining or fabricated department after the first operation is completed.
3. In a department when finished parts have been received from another department.
4. On the assembly line when the end product is complete.
5. In the shipping department when service parts, collaterals or help-outs are shipped.
6. In various plant locations when scrap, substitution or loss is reported.

The primary objective of MICS Phase II is to reduce parts shortages on the assembly line. MICS will also drastically reduce the expediting activity immediately before a tractor is threaded on the assembly line.

In the current semi-mechanical system, build tickets are examined the day before line thread by assembly line expediters. Because each tractor requires over 1,000 separate simple parts and assemblies on the line to complete its assembly, only the major units can be effectively analyzed by the expediter. Shortages on simple parts are sometimes difficult to determine. If shortages are discovered on major components, the production department must either reshuffle their build schedule or make an attempt to expedite within a few days the parts that are required.

Phase II of MICS compares the slotted order to the manufacturing authority build schedule. This will insure that all the materials have been ordered. When the slotted orders are within two to three weeks of the line thread date, the slotted order will then look ahead ten to fifteen working days to determine if finished components are available to build the planned machines. The projected parts shortages will be identified by their assembly line location. The shortages then can be expedited or the planned build schedule can be logically rearranged. This now allows considerably more lead time than existed prior to MICS.

A requirement to reslot a particular order can be easily analyzed. Then if this unit can be reslotted, the effect it will have on the other units, either in the pipeline or on the assembly line can be determined. Status of each specific order will be available to the Order and Distribution department which will provide better information for our Sales department and, of course, the customer.

The order slotting phase of MICS provides a considerable improvement in assembling the final machine. The logic of comparing the slotted build schedule against the availability of the simple parts and components would be almost valueless if the data provided in the first phase of MICS was unreliable.

MICS Phase III has three major sub-tasks—scheduling, order releasing and shop floor control. This phase adds new dimensions to our current system. Many of the techniques and business philosophies that have been built into our current batch material management system are being used in the new scheduling program. However, a radical departure from the old system is to base our orders on vendors and our manufacturing facilities on the available quantities shown on the stock status inventory generated in Phase I. The absolute necessity of reliable data is obvious.

The primary objectives of Phase III are:

to develop economic manufacturing quantities on all jobs released to the Melrose Park plant,
to reduce and even more important, to more appropriately allocate work-in-process inventories.
to reduce the amount of decision making by the first line supervision and replace these decisions with system generated schedules based on the broader scope of the total data, and
to improve the performance evaluation capability of the manufacturing departments.

In developing the concepts that will be used in this phase, extensive simulations and analyses of the data were made. The EMQ is developed at the high level assembly and works down to the simple parts. Therefore, because the lower level assembly is dependent on each subsequent assembly level, there are definite programming difficulties created in the calculation of EMQ's to fit at the various assembly levels. In many industries the cost of setup for assembly is low. However, in our case, this is not true. For example, major track weldments, rear frames and welded assemblies have significant setup costs. These setups combined with the low cost setups significantly complicate the programming logic when there is a schedule change.

In order to reduce the expediting required on a schedule change, a de-lotting routine was developed to minimize the effect of this change on our suppliers. Manufacturing can code a particular part as being critical. Tolerances will be established on this particular part number which coupled with the previous orders to that supplier will provide the necessary data for the system to adjust the order to the quantity to meet current production. This may be a non-EMQ quantity.

Phase III of MICS will develop infinite loading on all manufacturing facilities. This infinite machine load will reflect the total machine tool requirements but will not reshuffle or manipulate any overloaded conditions in the plant. This finite loading technique is being studied and will be developed in the future.

Extensive design and programming has been built into the system in an attempt to account for any changes made by de-lotting economic lots. This effort, however, will make the formal system more complete and, therefore, should eliminate the need for an informal or hot list system to circumvent the unreasonable demands for material.

A major requirement necessary for any online system is to have reliable data base files resident on a disc for access. Various techniques are available today for organizing data files on discs for rapid retrieval. Our experience started with the IBM Bill of Material Processor, then we tested extensively an IBM Users Bill of Material Processor, and finally decided to use the IBM Chained File Management System (CFMS).

The conversion of our data processing files to disc was complicated by the extensive integration of all our batch systems. Because we had developed major material management systems in the late '50s, file maintenance and report requirements were ingrained into these applications, and the extraction of portions or certain functions such as the file maintenance updating was, to say the least, extremely difficult. In many cases, these major applications had to be reprogrammed, or certain interfaces or bridges had to be constructed to support both the new and the old system. As time then would permit to redesign the application, these bridges then would be dissolved. The current manufacturing data base contains the following files: Product Structure, Piece Parts Master, The Bill of Material, Labor or Operation Routings, Work Center or Machine Tool File, a Location File, Employee File, Machine Asset File.

As a direct result of the MICS online data files and inquiry terminals, numerous other information files and sub-systems are being added presently or are being planned for the immediate future. For example, these include Assembly Error Analysis, Purchase Order Follow-Up System, Machine Order Processing System, Manufacturing Process Sheet System, and an expanded Man Data Base System.

Our confidence to go on-line through a remote processor was also based on the success we had with the IBM 370/155 under a multiprogramming environment. There are currently four high speed R.J.E. terminals operating in a HASP environment for data input, programmer testing and engineering testing. This computer is also used for the Corporate Message Switching System which has over one hundred terminals throughout the United States. A heavy load of divisional and plant batch applications are also processed. The I.B.M. 370/155 is also processing a similar on-line system for our Hough Plant 20 hours per day, six days a week.

Although the data collection control computer at Melrose Park, the IBM 2715, would be on-line 20 hours per day with the host computer at C.C.C., some buffering capability was essential in order to collect data during the off-line processing and during any unscheduled downtime. The IBM 2790 system, with its internal disk, would be able to buffer any telephone line or remote computer difficulty that would occur. The decision to go on-line to a remote computer was the determining factor on how the machine monitoring pilot department would be integrated into MICS.

The machine status and piece count data are collected by the IBM Systems 7. The IBM 1031 data collection units were replaced by the IBM 2791 units. All this data is consolidated in the host processor and, of course, is available for inquiry. The selection of the Systems 7 to replace the IBM 1800 was based on economics and its capability to expand the machine monitoring aspect of MICS throughout Melrose Park.

The overall system must be tailored to accomplish as many manufacturing objectives as possible. Then the hardware specifications can be written based on the performance parameter established in the original concept. If for reasons of cost or capability the equipment is not acceptable, the system must be restudied. This process will continue until an acceptable compromise between the objectives, costs and savings is achieved. The initial system design should never be centered around computer equipment.

Reliability and service support from the computer manufacturer were important considerations. The terminal units had to be able to interface easily with the host computer at C.C.C. Available computer software at the remote processor and the plant control computer were also major considerations.

The data collection units had to have some "intelligent" capability. This means, that the worker is led through his transaction by a series of lighted panels with verbal descriptions. If he enters an error, the terminal displays the error and "locks up" until he re-enters that step of the transaction. This step-by-step correction capability of the 2790 system considerably reduced the training and installation time. This overall capability of the data collection system was essential if the tight installation target dates were to be met.

Although the economics favored the IBM 2790 and 1050 systems, the advantages of having the same vendor's equipment at the plant and Computer Center are obvious. The complexities of installing an on-line manufacturing system would have been intensified if there were multiple vendors' hardware to install and evaluate during the installation period. Questions like, "Whose equipment or software caused the failure?" "Was the failure in the telephone line, modem or computer control unit?" would result in unnecessary delays. The vendor, IBM, assumed full equipment and software responsibilities as long as the entire MICS application from terminal to host computer was IBM equipment. This overall responsibility was a major consideration in our decision on equipment selection.

In the original design of MICS, every reasonable effort was made to solidify as many facets as possible in order to insure maximum reliability and rapid installation of the system. More sophisticated hardware and/or software technology that may have been desirable but not necessarily essential were deferred to more dependable and proven techniques. For example, although the 2790 system computer was not as flexible as the Systems 7, the Systems 7 software and hardware interface to the 2790 data collection units was not as well developed as it was for the 2790 control computer. Before the 2790 system was selected, programs were compiled and tested for the 2790 system computer. This early confidence in the 2790 system was later substantiated as this phase of the MICS program was installed and in production with absolutely no difficulties.

The initial feasibility called for IBM 1050 and 2740 typewriters rather than cathode ray tubes (CRT's) for several reasons. Although a CRT would considerably enhance the system to the user, the 1050 systems were considerably less expensive. However, the primary reason was the anticipated difficulties with the software to support the CRT's at that time.

A major decision without which the MICS system would not have been installed in such a short period of time was to program the entire on-line system in Cobol. Although many on-line systems are programmed in an assembler computer language to reduce core utilization and for generally more efficient processing, the development of the on-line programs in Cobol offered some distinctive advantages. Programming is considerably easier in Cobol than in Assembler. Cobol programmers are more readily available in the market and considerably easier to train. Our programming staff had been writing exclusively in Cobol since the mid-1960s. This in-house capability coupled with our projection of as many as thirty programmers required over certain time periods, clearly dictated that MICS be written in Cobol. Also, the plausibility of retraining our personnel in an assembler language for this project was questionable. The other advantages of less core utilization and faster throughput were generally neutralized by the relative low cost of computer memory on the IBM 370 computers and the low computer cycle utilization projected for our on-line programs.

The coordination for the installation of the communication lines and modems required at Melrose Park and C.C.C. was developed in detail with Illinois Bell Telephone. IBT provided special support and assistance in the final design of the communication network.

During all phases of MICS and particularly during the installation of the typewriters and data collection units during Phase I, in-depth planning and close coordination between Manufacturing, C.C.C. and Systems was paramount. Problem areas, regardless as to whose responsibility, had to be quickly resolved. Once MICS was in production in one department, any programming changes or equipment modifications had to be tested "off line" which generally meant Saturday or Sunday. This limited repair time reinforces the overall requirement to be as complete as possible in the detail system design. Because the Corporate Center operates 24 hours a day, seven days a week, any time required to physically install new equipment and make the necessary electrical changes had to be carefully scheduled, alternate plans had to be developed in the event the computer would not be functional on schedule.

A new employee badge system was also instituted. As a department went on data collection, each employee had to be given a new badge with his picture and pre-punched clock number. This, too, was a major coordinating effort. Special photographic equipment and badge materials had to be purchased.

The physical installation of the equipment and telephone lines and modems required extensive planning and coordination. The physical layout of the data collection lines in the plant had to be designed for backup if one unit in a department failed and also for easy installation and testing. The possible addition or movement of a terminal had to be anticipated.

The location of some of the typewriters/punch systems created physical problems because these units would not fit into many of the foremen's offices. They were either placed outside the office in a protected area or a special enclosure had to be fabricated to protect these units.

Now that the system has been in production for eighteen months, what are our plans to improve or strengthen MICS? Conceptually, the original design of the system has not changed. However, we are addressing hardware improvements designed to strengthen the up-time reliability and improved performance for the user. The 2715 control computer will be replaced by twin Systems 7's. Systems 7 will provide additional backup support, and also the capability of processing the machine monitoring phase of MICS. The division of the data collection loops between the two Systems 7's will provide additional backup within each department if one of the loops and/or Systems 7's should malfunction. The high speed printer and inquiry device in the MICS Control Room will be replaced by an IBM 3270 with a side printer. This will provide faster response to inquiries and give the option of printing to the individual making the inquiry. The 1050 inquiry typewriters will be buffered into the Systems 7. In the event the lines or computer malfunctions, the Systems 7 will determine this malfunction and message each terminal that the computer or telephone lines are not functioning and that their inquiries will be logged by the internal disk on the Systems 7 for transmission later. Also, several major systems will be changed to provide direct updating of the data base files used by MICS through IBM 3270 CRT's.

Obviously, there are many tasks that must be continuously monitored to insure the successful implementation of a manufacturing on-line system like MICS. Whether using a mechanical PERT or manual control system, some control mechanism must be employed. Not necessarily in order of importance, the following is a list of some of the more important aspects that must be addressed:

1. The system concepts must be developed by manufacturing personnel and supported totally by executive management.
2. A project manager must be assigned and should report directly to the highest level of manufacturing management in the division or works.
3. The data base should be analyzed in depth to determine its reliability before the project is approved.
4. A training program must include all levels of manufacturing personnel.
5. Computer equipment should be selected after the system is designed and not vice versa.
6. Equipment installation must be coordinated between the telephone company, computer vendor and the plant engineers.

MANUFACTURING MUST BE TOTALLY COMMITTED TO THE PROJECT AND MUST GET INVOLVED.

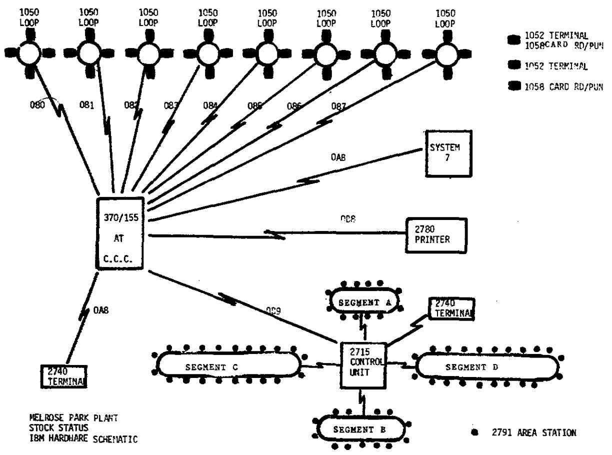


Figure 1

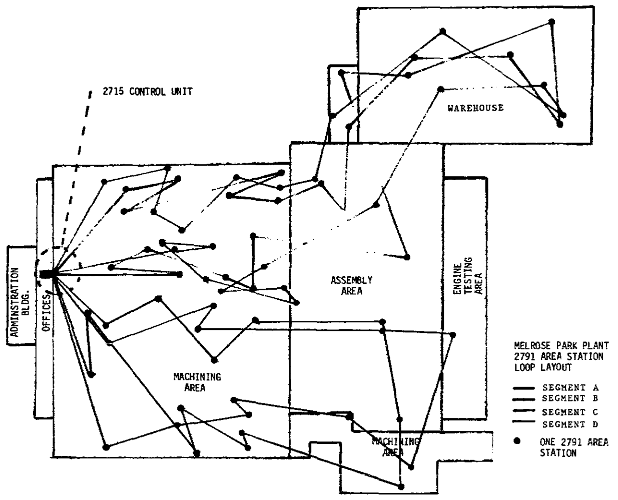


Figure 2

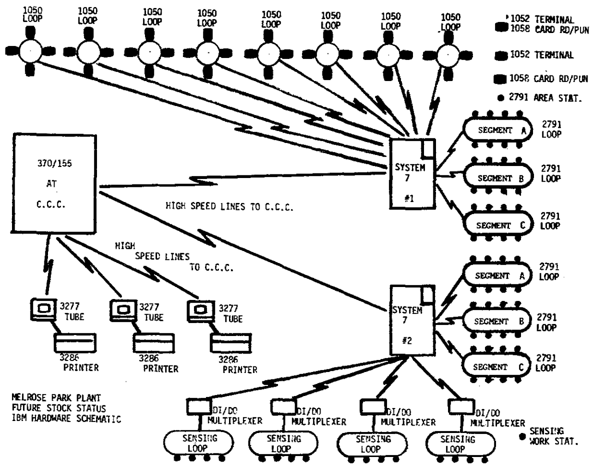


Figure 3

Remote data collection case study—Telephone Order Processing System (TOPS)

by M. H. RESNICK

Forman Brothers, Inc.
Washington, D.C.

INTRODUCTION

The IBM System/7 touch-tone order/entry inventory allocation system has been installed at Forman Bros. since January 1973. While the system is a Field Developed Program (F.D.P.) we at Forman Bros. feel we had a great deal to do with its development and implementation. The F.D.P. has been modified in the past year to fit our business requirements.

Early in 1972 we discussed the possibility of such a system with one of IBM's system engineers. From that point on, discussions became more lengthy and more serious. The F.D.P., which is now available, is a culmination of nearly nine months of systems and programming effort. We feel that it is a system which has wide applicability in the distribution industry.

We have prepared this brochure so that others may have an inside view as to how we justified needing the system, how we went about implementing the system, and how we justified its cost. We invite your comments and questions on our presentation and hope that this document will represent some helpful hints on how to use the system profitably.

FORMAN'S USE OF TOPS

In the District of Columbia, one of the nation's largest markets for the liquor distribution business, a slow answer about stock can result in the loss of a delivery day or the loss of a whole order.

With gross sales in excess of $28 million for the last fiscal year—a 45 percent increase since 1968—Forman Brothers found itself in need of a new way of processing its increasingly high volume of orders. Salesmen were calling their orders in to a tape recorder—telephone answering device, (Edison Phone System), but it took two hours after all of the orders were in for employees to keypunch, verify and edit them. There was also high keypuncher turnover and numerous keypunching errors.

Forman Brothers met the problem head-on by eliminating entirely its keypunching of salesman orders. Beginning in January 1973, it switched to a System/7 Order Entry/Inventory Allocation System. Each of the company's 42 salesmen were equipped with pocket-sized adapters for converting dial phones anywhere into Touch-Tone phones. Simply by keying in a customer code number and a product code, the salesmen communicate directly with the in-house System/7 as it allocates the inventory to each order and, just as fast, updates the inventory. At the end of each day, the disk pack from the System/7 is put into the company's general-purpose System/3 Model 10, which sorts the orders, produces invoices, and prints out management reports.

The salesmen can use all the time they can get. Forman Brothers serves some 1,200 retail outlets—including liquor stores, hotels, bars, restaurants, grocery stores, and markets—with a line of more than 3,000 different whiskeys, spirits, cordials, and wines. Invoices average 600 a day and it takes eight to 30 trucks daily to get the orders out. We average about 50,000 line items a month.

The System/7 knows, even to the last few bottles, what is on hand. If, for instance, an order comes in for a rare bottle of 1949 Lafite Rothschild wine, the computer will find it. Most of all, the retailers are happy because they now know exactly what they will be receiving.

Previously, the retailers did not know of any out of stocks until the trucks actually arrived the next morning.

The System/7 and System/3 go well together. For example, it makes no difference to the Model 10 when the System/7 receives an order. If two orders come in from the same customer during the day, the Model 10 will consolidate the order and produce a single invoice—and even will produce a route sequence for the trucks to follow in delivering the products.

Apart from the basic benefits of rapid and error-free order entry and inventory allocations, Forman's total system is being used to produce a number of reports of value to sales and company managers. Among these are sales reports showing what each salesman sold compared to quota, stock status reports, aged trial balances, and summarized warehouse location reports that are used to take a physical inventory of the company's entire stock.

WHAT IS T.O.P.S.?

TOPS is the most modern order processing system available in this country today. The computer hardware for TOPS is an I.B.M. System/7 Computer. The software consists of a set of programs developed by I.B.M., in conjunction with Forman Bros., to convert this System/7 Computer to an on-line, real-time, order processing system.

The system requires that salesmen enter their orders via a touch-tone telephone. In Washington, D.C. like most other areas, touch-tone telephones are not readily available in retail outlets or in public telephones. This is easily over-come by the use of a small, inexpensive, battery powered touch-tone adapter. The model we use is manufactured by Interface Technology, Inc., St. Louis, Missouri.

The system receives the orders from the salesman and does the following:

1. Edits the data entered by the salesman to determine that it is correct. That is to say, the salesman number, product number or customer number exists in our system. If this is the case, the computer will generate a tone to indicate this to the salesman. If, on the other hand, this data has been entered incorrectly, the computer will generate an error tone, indicating the salesman should re-enter the data correctly.
2. The computer automatically allocates inventory to each order as it is received, provided there is inventory available. If the order can be filled as requested, the computer so indicates to the salesman. If, on the other hand, there is an insufficient inventory condition existing, the salesman is informed and is given a choice as to what future action we should take. That is, cancel the order, back-order the entire order, or ship whatever is available.
3. The system has the ability to check each order against a pre-determined credit limit. If an order causes a customer to exceed this credit limit, the salesman is immediately informed of the situation.

   At Forman Bros., we do not use this feature of the system. Market conditions and credit regulations here are such that the value of this feature is too limited.
4. The system offers the salesman the luxury of an inquiry capability. That is, if a customer should ask "Do you have so many cases of a particular item?", the salesman can use the system to obtain an immediate answer. At the time he gets this response, he then has the option of cancelling the order or processing it. This allows our salesmen to offer a type of service never before available in our industry.
5. The system offers our management the most expedient means of contacting our salesmen. By placing an indicator in the salesman's file, he will be informed at the time he places his next order that he should call the office at the conclusion of his call.
6. The ability of this system to edit orders as they are received has had a major impact on our Data Processing Department.

   The very fact that the computer, working with the salesmen, has edited each order as it is received is a tremendous advantage in error reduction and/or elimination. We no longer have to key-punch orders at 5:00 p.m., key-verify them, edit them, correct them, and run the risk of making errors in the process of trying to correct errors. When a salesman completes his call, the order has been edited; it is correct; and it is in a form which can be immediately transferred to our billing process. There is no longer any intermediate steps or people required to prepare this order for invoicing.

   This error reduction benefit is such a tremendous time saving advantage, that we have extended the hours in which an order may be called in by two hours per day. We estimate that this is the time we have saved by not having to key-punch, key verify, and edit these orders.

To summarize TOPS: the salesman keys in orders through the telephone into the system. It, in turn, edits the orders, allocates inventory, checks for credit, and responds to the salesman telling him all he needs to know about his order. The system then prepares these orders for billing.

TOPS IMPLEMENTATION

In order to insure an effective, smooth, and error free implementation of the Telephone Order Processing System (TOPS), properly planned implementation is extremely important. At Forman Bros., we experimented with several different means. The following is a brief summary of the combined techniques we used to effect the kind of implementation which we desired.

The First Phase of implementation is to be done strictly by the Data Processing Department. In this phase, the Department must become completely familiar with the F.D.P., the way it works on the System/7, and the way the System/7 must interface with their host computer. In this phase, they must write and test those programs required to create the data files from the host computer and those programs required to take the data from the System/7 and read it back into the host computer. They must also check out all the additional management reports which are going to be required.

Within this phase, the Data Processing Department should also install the System/7, test all its programs, and be assured that all systems work as they are supposed to. It is extremely important, in this system as in any other, that the Data Processing Department have confidence in the system they are working with.

The Second Phase can begin mid-way through the first phase. This phase will require a close working relationship between sales management and the Data Processing Department. It is in this phase that the training technique for the Sales Department as well as the follow up in this training must be outlined and the details worked out.

In the Third Phase of the implementation, we divided all of our salesmen into small groups of 6 or 7 for the purposes of training. We also made a single group out of our sales managers. We then scheduled meetings with every Sales Department in two hour sessions for the purposes of indoctrinating them into TOPS. We chose to work with our sales managers first so that they would all know what was going on well before their salesmen did.

Having done this, we started with that group of salesmen who we thought would be the easiest to train, and then the next easiest, and so forth until we had scheduled all the various sales groups through the session.

In the first section of the training session, we gave the salesmen an historical background on the Telephone Order Entry Systems at Forman Bros. and the added benefits each one had over the previous one, until we finally were at the discussion of TOPS. The discussion of TOPS was merely an over-view at this point.

In the second section of the training session, we gave the salesmen a detailed, step by step instruction on how to use TOPS and what they should expect at each point along the way. We found visual displays and salesman involvement extremely helpful during this phase. The visuals enabled the salesmen to see what an order should look like, while their involvement helped overcome their fear and gain their interest. At the end of this session, we had a practice session for all salesmen supervised by people who were well qualified to answer all questions.

At the end of that session, each salesman was given a hand book with guide lines on TOPS explaining each field of data they would have to enter and each response they might receive at any point along the way.

The Fourth Phase was to be done by the salesmen on their own. This phase included numerous practice sessions which they could do in their spare time. When the system was not being used for teaching, we left it open for the salesmen who had been through the training sessions to use and practice on until they became confident that they could use the system quickly and easily. This training session was culminated with a quiz consisting of fill in the blank questions and short answer questions covering the general principles of the use of TOPS.

Having completed the quiz and being assured that we were on the right track, we were ready to begin our parallel operation of Phase Five. For this parallel operation, we took that group of salesmen which had performed best during our training sessions and had scored the highest on the quiz and picked two medium to heavy days of billing. We asked that these salesmen call their orders in both to the new system and the old system. We picked two days which would end on a Friday so that we would have the week-end to bill these orders, check them, and be sure that all systems were working properly.

Having done this, we began Phase Six, which was a slow but steady implementation of the new system by those salesmen who were qualified to use it. We started with small groups and had them use the system for all of their orders. We felt it was important to use small groups of people so we could monitor them closely and be sure that nothing went astray. We were concerned about both bugs in the system and errors that the salesmen might make.

This final implementation phase required a total of six weeks to complete. While most of the salesmen were ready and using the system three weeks after we started, we took it very slowly with those salesmen who were resisting the system for whatever their reasons. Finally, at the end of five weeks we held a final class and review session for these people and told them that at the end of the week, the old system would no longer be used.

Today, all of our salesmen use the TOPS System as their means of entering orders for next days deliveries (See Appendix A).

OPERATION NARRATIVE

Operation introduction

As the salesmen enter their orders through a 12-key touch-tone telephone or the TOPS adapter, they are guided through the order entry process. The system is open for the next day's deliveries between 8:00 am and 7:00 pm. The data is transmitted through a 403E6 data set into the System/7 disk for validation and subsequently, transferred to an order file located on both the fixed and removable disk packs.

Each day, the disk pack on the computer is initialized with an up-to-date inventory status. After the salesman has dialed the telephone number to connect him with the computer, he puts in a salesman security code, customer number, and delivery code to indicate when and where (front door, back door, side door, etc.) the delivery is to be made. He then enters the product number, quantity, and such additional information as special pricing and discount. For every field of data entered, the System/7 returns a "good" or "bad" beep tone. No invalid numbers will be accepted. If there is not enough inventory in the warehouse, a two-beep out-of-stock tone is given.

The end of an order is signaled to the computer by the salesman keying in a special action code. He can also tell the System/7 to accept a back order if there is an insufficient quantity on hand.

The order file, when placed at the end of the day on the System/3, will go through a detailed edit and reformatting procedure. The customer and product files are located in the same location but different disk drives. This shortens the seek time through reducing the arm movement of the disk.

In addition to the salesman placing orders, TOPS is used for entering credits and offers many inquiry capabilities.

Features and functions

There are many features incorporated in TOPS that allow the salesman inventory information, quantity limit warnings, backorder capabilities, field verification, end-of-order warnings, bottles greater than bottles per case entry warnings, no split case entry warnings, and zero quantity entry warnings.

Also included are displayed messages on the 5028 console indicating errors, out-of-stock conditions, and total quantity of orders entered.

The following are functions and tone generations:

01* END OF LINE ITEM
Sends you back to the product number
02* WHERE AM I?
This entry may be used at any time. A generation of tones will point to the next field of data to be entered.
03* ANY IN STOCK?
This may be used after a product number has been entered. It will let one know whether we have inventory or we don't.
04* ACCEPT PARTIAL BACKORDER ORDER
05* TEST
No files are updated—used for salesman training.
06* BACK ORDER ENTIRE QUANTITY
07* END OF LINE ITEM
Sends you back to the product number.
08* CANCEL ENTRY
An entire line item can be cancelled providing an 01* hadn't been entered.
09* END OF ORDER
Sends you back to the salesman field.

Operation

The following sequence must be followed in order to place a salesman's order:

Section 1 Field 1—Salesman

Salesman security number is entered and checked for verification on the removable disk pack. An invalid security number immediately disconnects him from the system.

Section 1 Field 2—Customer

The customer number is entered and scanned for verification on the removable disk pack. An invalid customer number will generate an error tone. If three errors occur, a rekey tone is generated and the salesman is disconnected from the system.

Section 1 Field 3—Delivery Override

Permanent delivery instructions are incorporated in the customer master file and would be used if there were no overriding instructions. The delivery overrides are contained in a table lookup with up to 99 entries. This is a two-digit number and any entry exceeding this will generate an error tone.

Section 2 Field 1—Product

The 2-5 digit product number is scanned for verification located on the fixed disk pack. If 3 errors occur a rekey tone is generated and the salesman is disconnected from the system. Each product number represents an individual size of a given brand.

Section 2 Field 2 and Field 3—Cases and Bottles

The case field may be skipped if bottles are to be ordered. Out-of-stock condition tones will be generated enabling the salesman to reply: cancellation 08*, partial order 04*, or full backorder 06*. Upon receipt of merchandise all backorders will be filled on a first-come-first-serve basis. Exceeding the quantity limit tone will be generated if quantities are greater than 10 cases. An exception table representing our fast moving and high volume brands does not generate a warning tone unless the limit exceeds 40 cases.

Section 2 Field 4—Discount Code or Amount

The discount field is optional. Standard discounts automatically apply to a product unless a discount code or amount is entered.

SYSTEM/3 PROGRAMS AND MANAGEMENT REPORTS

The System/3 programs needed for the interface of the System/7 are written in RPG II Language. The following programs are:

Load—This program formats the customer, product, and salesman file. Loaded are the current customer credit limit and balance, inventory balance, price per case, bottles per case, salesman sales number and security number. Exception tables for all files are loaded out to disk. Examples are: Salesman that are no longer with the company aren't loaded into the file. Customers that have changed ownership, but the salesman hasn't been notified, aren't loaded into the file. Products that we want to reserve for future sales aren't loaded into the file. Products that we don't split cases on are specially loaded into the file.

Edit—This program reformats and lists the System/7 files back into a readable format for the System/3 and proceeds through a more detailed edit check. Backorders are updated into the backorder file. Errors that weren't picked up from TOPS are denoted on the listing and a printed card of the error is produced and given to the salesman.

List—This program lists the order file in a convenient format, for use by the control clerk.

Salesman Usage—This program lists by salesman number all orders placed, the time, and how many line items and cases were called in. This is used by TOP management to analyze a days business by a salesman.

Backorder—This program lists all backorders that can be filled or partially filled. It will only automatically fill the backorder upon receipt of merchandise if there is an existing order going to that account. If not, the backorder is given to our customer service department for further processing.

Billing—The invoice denotes whether or not the order was placed by a salesman, customer service department representative, or by a computer filled backorder.

Sales Report—This daily report is given to each salesman showing him what orders were both placed by our customer service department or from TOPS. It shows every detailed line item invoiced or backordered (See Appendix B).

OPTIONAL INQUIRIES

QNT

Anytime one wishes to know total cases and bottles ordered through that time. Warehouse uses this for estimating the number of trucks.

ITM

Used for inquiring to the Product Master File located on the fixed disk, and used to alter any of the following fields:

Item Number
Inventory On Hand In Bottles
Order For The Day In Bottles
Backorders For The Day In Bottles
Bottles Per Case
Price Per Case
Case Quantity Limit

CST

Used for inquiring to the Customer Master File located on the removable disk, and used to alter any of the following fields:

Customer Number
Credit Limit
Balance In His Account

SLM

Used for inquiring to the Salesman File located in core, or used to alter any of the following fields:

Salesman Security Number
Set A Call In Tone By Setting Bit On
Salesman Sales Number

DMP

Used to dump certain blocks of information.

$UDPAT

Used to patch instructions on to disk.

PAT

Used to patch instructions into core.

BENEFITS AND JUSTIFICATION

Orders are processed in such a way as to insure fewer errors. The salesman is the only person involved in processing his order through the computer. There is no key-punch operator who is subject to making mistakes. There is no editing that may result in incorrectly correcting an error. It is the salesman's responsibility to see to it that the information given the computer is correct. We feel that this combination of putting the responsibility on the salesman, while at the same time eliminating all other people from the process, results in fewer errors.

The system provides up-to-the-minute inventory information. This allows the salesman to inform his customer specifically of what quantity of merchandise is available of any given item, at any particular point in time. He can, at the same time, assure the customer of delivery. The system, you will recall, allocates inventory for each order processed and, therefore, each salesman is entering orders against the most current inventory available.

Existing out-of-stock conditions are known at the time the salesman places an order. He can immediately recommend alternatives to his customer or take alternative steps on his own.

Our salesmen's ability to pass on immediate out-of-stock information to a customer results in a better customer/salesman relationship. Customers no longer are disappointed, having ordered a particular item only to find, at the time of delivery the next day, that it is out-of-stock. They know that unfortunate condition at the time the order is placed and will be able to consider alternatives and make their plans accordingly.

This current inventory information can help our customers better service their customers. In the case of specialty items, they can know, at the time they place an order, whether or not it can be filled. At the same time, in general, they are able to insure a better selection of products to their customers.

Under our old system we spent approximately two hours after 5 o'clock key-punching, key verifying, and editing orders in preparation of running invoices. This involved process was eliminated by this system. The orders, once received by TOPS, are in a form which can be directly passed into the computer. There is no key-punching, no key verifying, and no editing.

This benefit allowed us to extend our business hours by a minimum of two hours each day. Instead of cutting off orders for next day delivery at 5 o'clock, we extended that time to 7 o'clock. We feel that the salesman's ability to offer this last-minute service to our customers is a great benefit to all. The salesman is able to promise orders for next day delivery more often to more customers. At the same time, he can eliminate those will-takes which are now required to insure that kind of service. Over all, we have fewer exception type invoices than we did before.

We can put a message on the system requesting that a particular salesman call the office at the end of that particular call.

As indicated earlier, the system was designed to be used by the salesman, in the field. Through experience, our salesmen found that this technique is not always the most desirable.

Many of our salesmen prefer to work through the entire day and call their orders in at the end of the day. They feel that working in this way increases their efficiency and gives them more selling time. (This does not interfere with our scheduling because the System/7 eliminates key-punching log jams 100 percent.)

The above represents those areas which we see as major benefits. There are, of course, other benefits to be had from the system but they will vary considerably from one company to another.

In the same way, justification will vary considerably from one company to another. We will, therefore, merely highlight the most important areas to be considered.

The first consideration is the cost of the system. Costs may vary from one location to another, so we will refrain from quoting specific dollar figures. Your list should, however, include the following items:

1. Rental of IBM System/7.
2. Rental/Purchase of the IBM Order Entry/Inventory Allocation Program.
3. Rental of all necessary telephone equipment.
4. Purchase of telephone adapters—if necessary.

While this list of monthly expenditures may represent a considerable amount of money, the system does offer the potential for a considerable reduction of current costs. The following areas represent potential cost savings:

1. The number of people on the key punch staff who are currently required to key punch and key verify those orders written and called in by salesmen may be eliminated. (Retain those people required to process other kinds of orders and other kinds of work.)
2. Depending on the individual company's procedure, it may be possible to significantly reduce current expenditures in the order processing department. People who are used primarily for coding orders or scan sheets will not be required.
3. Further cost reductions will be realized through the elimination of key punch machines and other order processing equipment.

Overall, a company using an IBM System/3 Model 10 will realize a true cost saving if they can eliminate two (2) key punch operators and two (2) key punch machines.

In addition to the potential of a true dollar savings, the system offers important money saving and/or profit making opportunities in the areas of:

1. Increased salesman selling time.
2. Reduced warehouse shipping time (invoices can be available sooner).
3. The value of the other benefits already mentioned.

IDENTIFY ANY SIDE EFFECTS OF THIS SYSTEM AND SHOW HOW TO TAKE ADVANTAGE OF THEM

There are several areas where side benefits could be attained from this system. First is the potential use of the System/7 in other areas. Among these are warehouse control and accounts payable. These two areas are currently being investigated by Marty Resnick.

Secondly, the very concept of using an order entry system which offers response to the salesmen, will build their confidence. Under this system, they will not be completely in the dark as to whether or not the system is working and receiving their orders properly. The system, as you will recall, gives a response to everything the salesman does—right or wrong. He always knows that the system is working and has received his order. He will not, of course, be talking to an individual, but he will have a better feeling than he does today talking to a tape recorder.

Another side benefit in error reduction will be the elimination of records. The records we currently use in our Edison Phone System, can be handled improperly and hence, errors can result. A record can be put on an Edison machine and be recorded on twice, as well as being entered into the computer twice. Clearly, we take steps to avoid this, but also, we must be aware that it can happen. The fact that the computer retains the orders internally eliminates this potential danger area.

Finally, and perhaps most importantly as a side benefit is the potential this system offers for notoriety for Forman Bros. It is IBM's intention to market this system on a nationwide basis. As such, we will be the first working installation. We have already received calls from many people around the country.

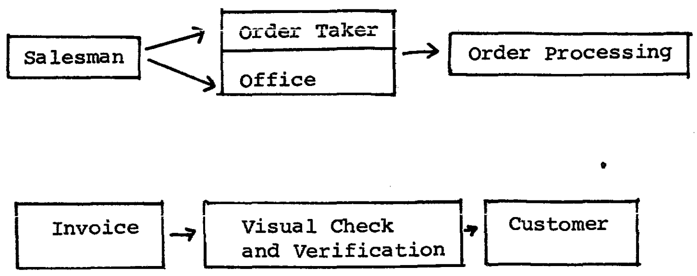
At the same time, it will be in our best interest to show this system to suppliers when they come to visit, letting them know how advanced we are. We feel that everyone will be impressed by the use of this system as a sales tool. We know, from what little experience we've had, that people who are sales oriented are quick to appreciate the benefits of this system.

HISTORICAL BACKGROUND

Pre-Code-a-Phone

Prior to 1965, our order processing system was operated without the aid of any electronic devices. Most orders were brought back to the office by the salesmen late in the afternoon. If this was not possible, the salesmen would call orders into an order taker.

Listed are the 7 steps needed to process the order:



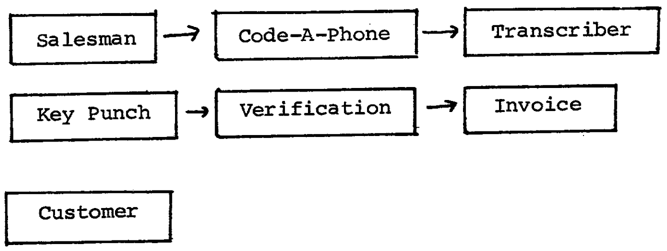
Code-a-Phone

About 1965, we introduced our first electronic aid to order taking. The code-a-phone allowed the salesmen to call their orders into the office at any during the day up to 4:00 pm.

The salesman was freed of the burden of bringing orders back to the office. At the same time, the order takers had more time to spend with customers.

One major draw-back, however, was that many people had to handle the order. This created many opportunities for error.

Listed are the 7 steps needed to process the order:

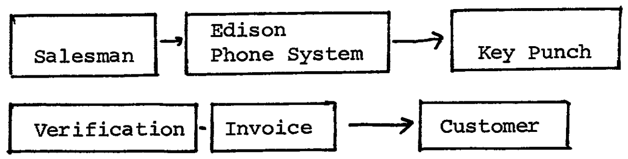


Edison Phone System

In 1970, we took the next step toward refining our order processing system. The Edison Phone System allowed us to by-pass the transcription step in order processing. By key punching directly from the salesman's record order, we accomplished two important goals:

1. We eliminated one area in which errors could be made.
2. We realized a time savings that allowed us to extend the time in which orders could be called in by 5:00 pm.

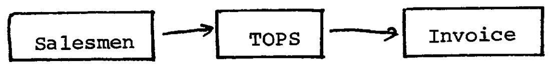
Listed are 6 steps needed to process an order:



T*elephone* O*rder* P*rocessing* S*ystem* (TOPS)

We have now completed the next step in the process of refining our order entry system. TOPS will speed order processing, eliminate unnecessary errors, and provide the salesmen with more information about inventory availability.

Listed are 3 steps to process an order:



For the first time since we introduced electronic order taking aids, the salesman will get feed-back, through the system, to advise and guide him in placing an order.

FILES

SALESMAN—contains 99 records
CUSTOMER—contains 10,000 records
INVENTORY (ITEM)—contains 10,000 records
ORDER—contains 25,000 records
Every order produces:
1. Regular Record
2. Statistics (Time, total errors and line items)
3. End of order record
4. Dummy record

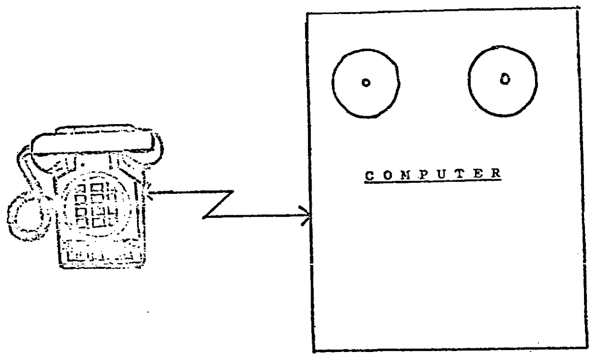
TRANSACTIONS

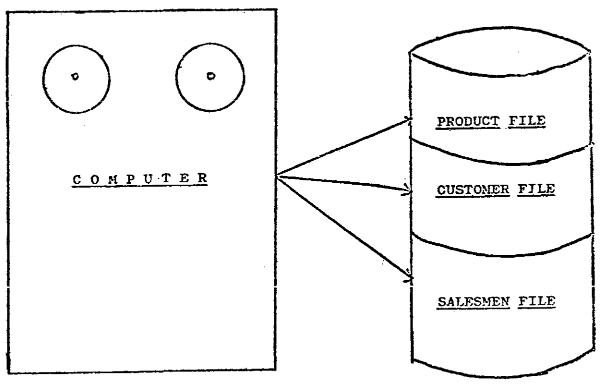
01*	END OF LINE ITEM
02*	WHERE AM I?
03*	ANY IN STOCK?
04*	ACCEPT PARTIAL ORDER
05*	TEST—NO FILE WRITE
06*	BACK ORDER ALL
07*	END OF LINE ITEM
08*	CANCEL ENTRY
09*	END OF ORDER

FIELDS

1. SALESMAN SECURITY NUMBER
2. CUSTOMER NUMBER
3. DELIVERY OVER—RIDE
4. ITEM NUMBER
5. CASES
6. BOTTLES
7. DISCOUNT
8. INVOICE NUMBER
9. SPECIAL PROMOTION NUMBER
10. OPTIONAL
11. OPTIONAL
12. OPTIONAL

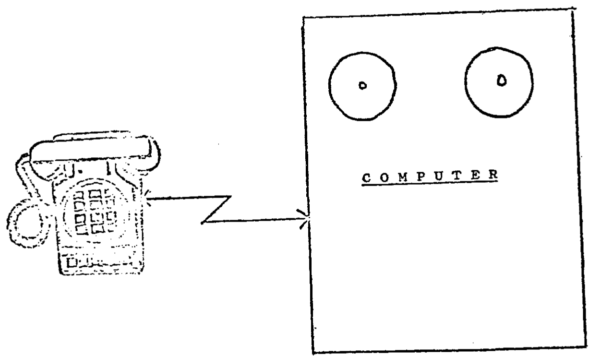
APPENDIX A—ORDER FLOW



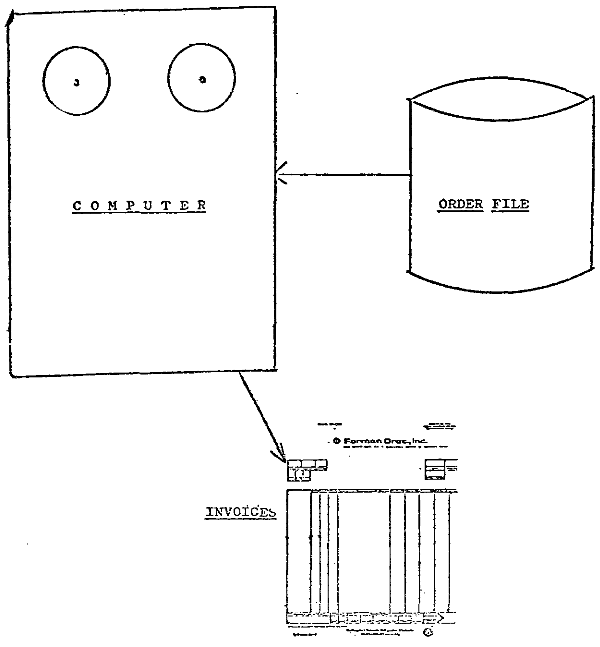


SALESMEN WILL USE A TOUCH - TONE TELEPHONE TO KEY ORDERS INTO THE COMPUTER.

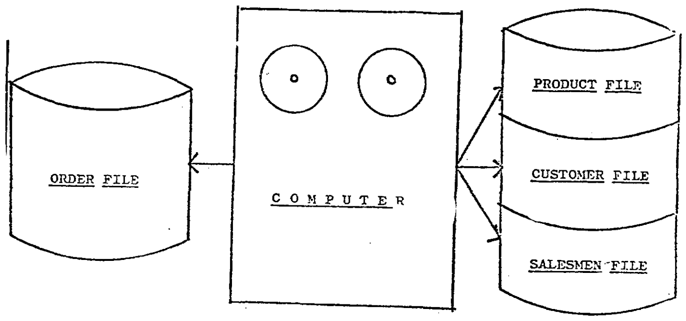
THE COMPUTER WILL CHECK THE ORDERS AGAINST THE THREE MASTER FILES.



IF ERRORS ARE DETECTED, THE SALESMAN WILL BE NOTIFIED BY THE COMPUTER.



AT THE END OF THE DAY, THE ORDERS ARE PROCESSED BY THE COMPUTER AND INVOICES ARE PRINTED OUT.



IF NO ERRORS ARE DETECTED, THE COMPUTER WILL UPDATE THE APPROPRIATE FILES AND RECORD THE ORDER IN THE ORDER FILE.

APPENDIX B

```
 SYS02              ECIT CRCERS ANC CRECITS      12/12/73


CLST SLSMN CS BCT PRCC# C/CCE C/AMT    CEAL  CS BOT    AMT  INVCICE#
13112 37    1      6257   3    2.5C
16C05 03    5       127   3   15.5C
13112 37    1      6178   3    2.5C
16005 03    5       141   3    4.63
13112 37    1      6226   3    2.5C
13112 37    1      6374   3    2.5C
16C05 03    1     33158   1
13112 37    2     13875   1
16005 03    1       983   3    5.45
13112 37    1     13882   1
13112 37    1     139C9   1
16005 C3        4 11866
13112 37    1     51408
 5373 37    1     75936
16C05 03    1        24
 5373 37    1      1795
16005 03        6  55CC   3    5.25
 5373 37    1     762C5
16005 03        6  5555   3    5.25
 5373 37    2     13875   1
16C05 C3        6  5799   3    5.25
 5373 37    2     139C9   1
 5373 37    1      1379
16005 C3        6  5854   3     .25
 5373 37    1      1386
 5373 37    2      1489
16C05 03    2     57868   3    5.5C
 5373 37    1      1496
 5373 37    1      1290
16C05 03    1     9988C                Y41    1      13.86
05373     001 000 834C7           BACK CRCER
16C05 03    1     99880                Y33    1       1.CO    60315
16C05 03    5     99880                Y29    5       1.5C    57562
16C05 C3    5     9988C                Y29    5       2.CO    57562
16005 03    1     99880                Y41    1      17.C1
8C002     000 006 73037           BACK CRCER
8C002 55        6 73037
8C002 55        3  9669
8CCC2 55        1  9621
8CCC2 55    2       213   1
8CCC2 55    1       22C   1
8CCC2 55    1       237   1
8CCC2 55       12  3917
464C4 55    1     20235
46404 55    1     762C5
464C4 55        2 14773
464C4 55    2       2C6   1
464C4 55    1       213   1
46404 55    1       22C   1
464C4 55    2       237   1
 31C6 55        6 2C417
426C6 55    1     73075   1
42606 55    1     73044   1
46404 37    1      1764
426C6 55    1     73051   1
```

```
OEFDP CL     24930    16305
OEFDP 52     22639    34719      141
OEFDP CL
OEFDP CL     24930    62374
OEFDP CL
OEFDP 42     21834    69450    99770
OEFDP 42     21834    69450    99770
OEFDP CL     24930    29203
OEFDP 10        77
OEFDP CL     23630    69656
OEFDP 42     21834    69450    99770
OEFDP CL     22639    34719    99770
OEFDP CL     23630    48808
OEFDP CL     21834    69450    99770
OEFDP 62     21834    36560      804
OEFDP CL     22639    68806
OEFDP CL     21834    36560       31
OEFDP 62     21834    67805      354
OEFDP 52     23630    37059     4925
OEFDP 52     22639    69508      141
OEFDP 42     21834    67805      141
OEFDP 40     22639    69508     3315
OEFDP 52     21834    67805      141
OEFDP CL     23630    37059
OEFDP 53     24631     6604    80332
OEFDP CL     24631     6604
OEFDP 40     22639    69508    97793
OEFDP CL     23630    24356
OEFDP CL     24631    73707

OR1:QNT
   1087 CASES         970 BOTTLES

OR1:OFF
OEFDP CL
OEFDP CL     22639    69508    99770
OEFDP CL     21834    67805    69230
OEFDP CL     22639    72218
OEFDP 10    226399
OEFDP 52     24631    17903      574
OEFDP 52     21834    69601      141
OEFDP 52     21834    69601    16593
OEFDP 52     24631    17903      141

OR1:QNT
   1176 CASES        1040 BOTTLES
OEFDP 89     24631    17903
OEFDP 62     21834    69601     8228
OEFDP CL     21834    69601    99770
OEFDP 10        77
OEFDP CL     24631    17903
OEFDP 52     24631    52203    33282
OEFDP 52     24631    52203    33309
OEFDP CL     24631    52203

OR1:QNT
   1220 CASES        1116 BOTTLES
OEFDP CL     24631    58601
OEFDP CL     24631    42606
OEFDP CL     24631    62501
OEFDP CL     24631    55220
OEFDP 52     24631    46033      574
OEFDP CL     24631    46033
OEFDP 10        77
   1252 CASES        1136 BOTTLES
SYSTEM IS SHUT DOWN
  TIME 17:16
```

```
      12/12/73              CRCER FILE LISTING                                                4

                                                       ------------ CEALS  ------------      -TIME-
 SLS   CUST   PRCD.  -CRCER-  -BK CRC  ---CISC-----    AC      INVNC  CS  BCT    AMT     CC  H  M S
                      CS  BCT  CS  BOT  CC     AMT
  60  19400    684       24                                                                 15-46-12
  88  68837    787        3                                                                 15-46-13
  88  68837    512        3                                                                 ~~15-46-20~~
  88  68837   START   15H45M24S  ENC 15H46M2CS  ITEMS       6  ERRCRS        1             TCTAL AMT

  6C  19400  81883    1              3       9.8C                                           ~~15-46-20~~
  60  19400   START   15H45M59S  ENC 15H46M2CS  ITEMS       2  ERRCRS                      TCTAL AMT

  88  77880    512        3                                                                 ~~15-46-33~~
  60  17996  20084        6                                                                 15-46-38
  88  77880   4705        3                                                                 15-46-41
  88  77880   START   15H46M22S  ENC 15H46M43S  ITEMS       2  ERRCRS                      ~~TCTAL AMT~~

  60  17996    206    1                                                                     15-46-43
  60  17996    220    1              2                                                      ~~15-46-49~~
  60  17996    237    1              2                                                      15-46-55
  88  11093    172    1              3       8.CC                                           15-47-0C
  88  11093    134        8                                                                 ~~15-47-05~~
  88  11093   START   15H46M45S  ENC 15H47MC5S  ITEMS       2  ERRCRS                      TCTAL AMT

~~60  17996  75101    1              3       3.5C                                           15-47-13~~
  6C  17996  75259        2          3       1.75                                           15-47-21
  60  17996  20118        1                                                                 15-47-26
  6C  17996   9346    1              3       1.5C                                           ~~15-47-36~~
  60  17996  16823    1              3       1.5C                                           15-47-45
  6C  17996   START   15H46M23S  ENC 15H47M59S  ITEMS       9  ERRORS        2             TCTAL AMT

  60   7557  69735        2          3       5.2C                                           15-48-15
  60   7557  47120        2          3       5.2C                                           15-48-22
  60   7557  20118        1                                                                 ~~15-48-28~~
  60   7557  10243    2              3       9.95                                           15-48-41
  60   7557    660    2              1                                                      15-48-48
  60   7557    677    1              1                                                      ~~15-48-54~~
  60   7557    213    2              1                                                      15-49-00
  60   7557    220    2              1                                                      15-49-C6
  29   4705    763    1                                                                     ~~15-49-07~~
  60   7557    237    2              1                                                      15-49-12
  29   4705    251    1              3      12.49                                           15-49-14
  6C   7557    196    1              1                                                      ~~15-49-18~~
  29   4705    275        6                                                                 15-49-21
  60   7557    206    2              1                                                      15-49-24
  54  70807    220    2              1                                                      ~~15-49-31~~
  60   7557   9951    3              3       8.8C                                           15-49-32
  29   4705    543    1                                                                     15-49-34
  54  70807    677       12          1                                                      ~~15-49-38~~
  6C   7557  82774        9                                                                 15-49-39
  60   7557   9944    1              3       6.5C                                           15-49-46
~~54  70807  81144               6   1                                                      15-49-55~~
  60   7557  99770                           1.7C     264          1                        15-49-56
  54  70807   START   15H49MC7S  ENC 15H49M57S  ITEMS       3  ERRCRS                      TCTAL AMT

  29   4705    127    1              3      15.5C                                           15-49-58
  60   7557  99770                           2.65     264          9                        15-50-C8
  60   7557   START   15H48MC3S  ENC 15H5CMC8S  ITEMS      16  ERRCRS                      TCTAL AMT 1,5
```

```
DATE 12/12/73        SALESMAN S USAGE OF TOPS

SALESMAN  ROD WILLIAMS

 CUSTOMER                                      TIME          NO OF ITEMS
HOLIDAY LIQUOR                                16-58                    5
LOG CABIN LIQ STORE                           16-54                    8
OCEAN VIEW LIQUOR                             16-55                    2
SHIPLEY BEVERAGES                             16-59                   18
SOUTH CAPITOL LIQUORS                         16-56                   12
SPAR LIQUORS                                  17-01                   32
 TOTAL CUSTOMERS        6                                  TOTAL      77
```

```
DATE 12/12/73                  BACKORDER STATUS
PROD. DESC.                  CUST  DO  CS  BOT  --DISC--    DATE    STANDING

00141 J & B SCOTCH           22323 00 007 007  3   4.63   12/12/73   ***** FILLED *****
00141                        01692 00 010 000  3   4.63   12/12/73   ***** FILLED *****
00141                        41605 00 005 000  0          12/12/73   ***** FILLED *****
00141                        34719 00 015 000  3   4.63   12/12/73   ***** FILLED *****
00141                        69508 00 004 000  1          12/12/73   ***** FILLED *****
00141                        67805 00 003 000  0          12/12/73   ***** FILLED *****
00141                        69601 00 005 000  0          12/12/73   ***** FILLED *****
00141                        17903 00 010 000  1          12/12/73   ***** FILLED *****
00354 BBN SUPREME            52203 00 000 039  1          12/11/73   ***** FILLED *****
00354                        65793 00 000 012  0          12/11/73   ***** FILLED *****
00354                        52203 00 001 000  0          12/11/73   ***** FILLED *****
00354                        47450 00 000 012  0          12/12/73   ***** FILLED *****
00354                        01692 00 001 000  0          12/12/73   ***** FILLED *****
00354                        34719 00 002 000  1          12/12/73   ***** FILLED *****
00354                        67805 00 000 012  0          12/12/73   ***** FILLED *****
00354                        21504 00 001 000  0          12/11/73   ***** FILLED *****
00488 MEECWOOD 100           20819 00 000 023  0          12/04/73   ***** FILLED *****
00488                        78300 00 001 000  0          12/05/73   ***** FILLED *****
00488                        01692 00 001 000  0          12/12/73   ***** FILLED *****
00488                        59114 00 001 000  0          12/12/73   ***** FILLED *****
00488                        20448 00 001 000  0          12/05/73   ***** FILLED *****
00488                        78513 00 001 000  0          12/05/73   ***** FILLED *****
00488                        21700 00 001 000  1          12/05/73   ***** FILLED *****
00488                        69625 00 001 000  0          12/05/73   ***** FILLED *****
00488                        46538 00 000 012  0          12/06/73   ***** FILLED *****
00488                        06202 00 001 000  0          12/08/73   ***** FILLED *****
00512 DRAMBUIE               79019 00 001 000  0          12/11/73   ***** FILLED *****
00512                        80112 00 001 000  0          12/11/73   ***** FILLED *****
00512                        23403 00 000 002  0          12/11/73   ***** FILLED *****
00512                        70704 00 001 000  6          12/11/73   ***** FILLED *****
00512                        24923 05 000 003  0          12/11/73   ***** FILLED *****
00512                        27108 05 001 000  0          12/11/73   ***** FILLED *****
00512                        50809 00 000 003  0          12/11/73   ***** FILLED *****
00512                        12276 00 000 006  6          12/11/73   ***** FILLED *****
00512                        01764 00 000 006  5          12/11/73   ***** FILLED *****
00512                        14807 00 022 000  3  13.09   12/11/73   ***** FILLED *****
00512                        79150 00 000 001  0          12/11/73   ***** FILLED *****
00512                        59835 00 001 000  0          12/11/73   ***** FILLED *****
```

```
01960 CCRN STILLBRCCK      C1764 CO CCC CC5  C         12/C7/73   ***** FILLED *****
04925 KAHLUA               59217 C1 CO1 CCC  C         12/C7/73   ***** FILLED *****
04925                      29454 CC CC1 CCC  C         12/C7/73   ***** FILLED *****
04925                      23104 1C C1C CCC  C         12/1C/73   ***** FILLED *****
04925                      23104 CC C1C CCC  5         12/1C/73   ***** FILLED *****
04925                      C9913 CO CO3 CCC  5         12/1C/73   ***** FILLED *****
04925                      77488 CO CO1 CCC  C         12/1C/73   ***** FILLED *****
04925                      70704 CC CC1 CCC  6         12/1C/73   ***** FILLED *****
04925                      29708 CC CC1 CCC  5         12/1C/73   ***** FILLED *****
04925                      C6910 CO CCC CC3  C         12/1C/73   ***** FILLED *****
04925                      234C3 CC CCC CC1  C         12/11/73   ***** FILLED *****
04925                      15509 CC CC1 CCC  C         12/11/73   ***** FILLED *****
04925                      11505 CO CCC CC2  C         12/11/73   ***** FILLED *****
04925                      17501 CC CCC CC3  C         12/11/73   ***** FILLED *****
04925                      C0907 09 004 CCC  C         12/11/73   ***** FILLED *****
04925                      C09C7 C9 CC4 CCC  C         12/11/73   ***** FILLED *****
04925                      C5373 CO CO1 CCC  C         12/11/73   ***** FILLED *****
04925                      C6532 CC CCC CC5  C         12/11/73   ***** FILLED *****
04925                      51659 CC CCC CC1  C         12/11/73   ***** FILLED *****
04925                      34908 CC CCC CC6  C         12/11/73   ***** FILLED *****
```

APPENDIX C—FORMAN BROS., INC.—"TOPS" (TELEPHONE ORDER PROCESSING SYSTEM)—SALESMAN TRAINING AND REFERENCE MANUAL

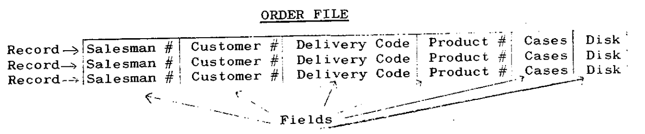
DESCRIPTION AND USE OF THE DATA FIELDS

Definitions

FILE: A complete list of similar types of data.
Ex: Customer file—All customers
Product file—All products
Salesman file—All salesmen
Order file—All orders

RECORD: All the elements of data related to a particular entry in a file.
Ex: A customer record
A product record
A salesman record
An order record

FIELD: A single element of data in a record.
Ex: In a customer record, the customer number is a field.
In a product record, the price per case is a field.
In a salesmen record, the total calls made is a field.
In an order record, the case quantity ordered is a field.



Salesman security number

The salesman security number represents your salesman number and other incorporated numbers acting as a security code.

This security number is built into the "TOPS" system to protect your orders. It belongs to you only, and it should not be given out to anyone.

Customer number

This number represents the account number that is associated with one individual customer.

Delivery override number

The delivery override represents the delivery instructions that will override the permanent delivery instructions when the invoice is printed.

The salesman has the option to ignore this field if he has no special delivery instructions. You then must key in an * to bypass the field.

Product number

This number represents the product number that is associated with each individual size of that brand.

Case quantity

This number represents the number of cases that has been keyed in. You may key in up to a three digit number. Example: 100.

Bottle quantity

This number represents the number of bottles that has been keyed in. You may key in up to a three digit number. Example: 180.

Discount code or amount

This number represents either the discount code or the discount amount.

The discount codes used may be:

A. Code 1—Monthly Promotion (deal of the month)
This should only be used if your salesmanager has asked you to.
B. Code 2—Standard High Discount
This should only be used if your salesmanager has asked you to.
C. Code 5—Net Item
D. Code 6—All discounts on every line item will be net.

The discount amount should only be used if the standard normal discount is different, or if the discount code 1 or 2 is not applicable. You may key in up to a four digit number. Example: 1000 (which is $10.00) *YOU MUST USE DOLLARS AND CENTS*!

Deal type number

This number represents the *type* of deal that is going to apply. "Discount Due" is now represented by the number "99770." "Completion of A Deal" is now represented by the number "99880."

Deal cases

This number represents the number of cases on which you are giving a discount.

Deal bottles

This number represents the number of bottles on which you are giving a discount.

Unit discount factor

This number represents the discount factor that will be multiplied by the deal cases and/or deal bottles.

Invoice number

This number represents the invoice number to which reference is being made as to why the discount is being applied. This should always accompany a discount due or completion of a deal.

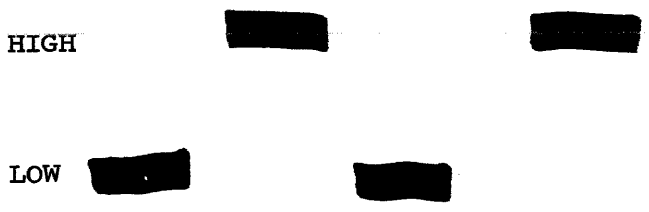
Deal number

The deal number pertains to a specific product that the deal discount is applying. This is a two position number and should be used for that specific brand only.

This must always be used when giving a "deal discount."

ADDITIONAL SALESMAN OPERATING ENHANCEMENTS

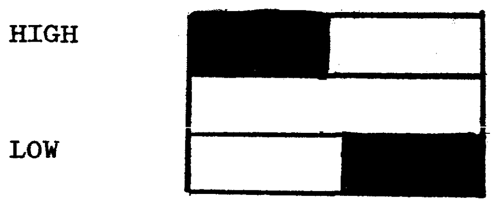
1. Case quantity limits are incorporated into the TOPS System. Should you order an item in a quantity greater than 10 cases (or greater than 40 cases for those items which have been designated as exceptions) you will receive a warning tone.
   When you receive this tone, you have two options available:
   a. If you order exceeds the limit (and you are sure you called it in correctly), continue as though you had received a proceed tone. We will accept and bill whatever quantity you entered.
   b. Cancel the order (08*) and re-enter the order correctly.
2. A warning signal will occur if the bottle quantity you ordered exceeds the number of bottles per case. At this warning tone, you must cancel the order with an 08* and rekey that product with the correct bottle quantity.



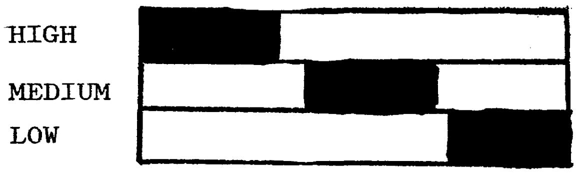
3. A warning signal will occur if you keyed in a bottle quantity that is a no-split case item. Examples of these items:

   Yago Santgria
   Gallo
   Wild Irish Rose
   Bartenders

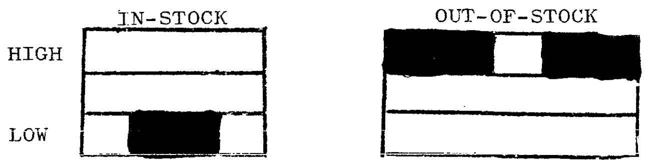
   At this warning tone, you must cancel the order with an 08* and rekey that product with the correct case quantity.
4. 07* has been added to serve the same function as the 01* (end of a line item). The 0, 7, and * are all closely accessible to each other on the telephone. Another reason for this addition is the fact that a large percentage of our orders are called in units of 1, causing temporary confusion as to where you are.
5. The 01*, 09*, and 07* will now generate a different tone.



   This was added to distinctly signal the end of a line item or completion of a customer order.
6. During a regular order, it is now impossible to go beyond the discount field. A call-in tone will be generated, clearing out the line item and placing you back at the product field.

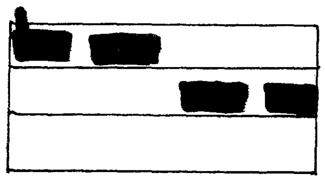


   At the sound of this tone, it will be safer for you to key in your last 3 line items for that customer or call the Computer Department for assistance.
7. The insufficient quantity tone has been changed so that you will receive only 2 tones when an order cannot be completely filled.
8. The only temporary confusion you might encounter will be when keying in an item greater than 10 cases (or greater than 40 cases for those items which are designated as exceptions), and receiving an insufficient quantity tone. After answering a 04* or 06*, you will receive a warning signal meaning this item has exceeded the quantity limit.
9. The transaction code, 03*, is used to seek information on a product as to whether or not we have quantity or no quantity. If there is inventory on hand, it will give you a proceed tone. If there is zero inventory, you will be given a normal insufficient quantity tone.



You may use this code after a product number has been keyed or after you have received an insufficient quantity tone. In either use, TOPS will send you back to the product field.

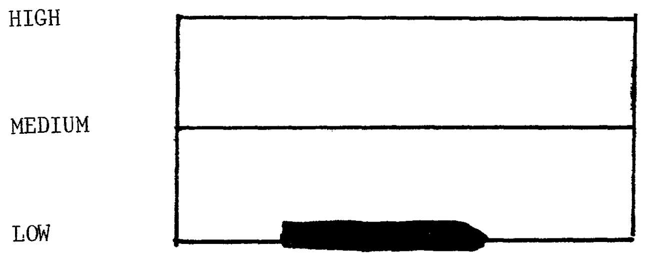
10. If you go too fast and fail to enter quantity you will receive a zero quantity tone. At the sound of the tone you will be in the product field and ready to reenter that item again.



RESPONSE PROCEDURES

Proceed tone

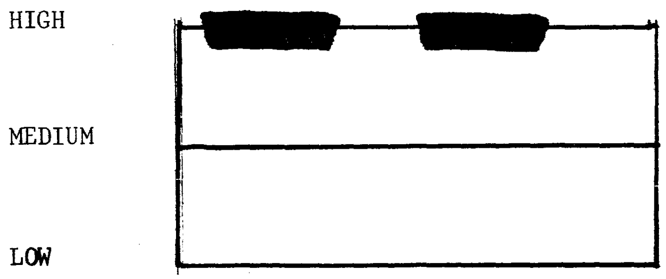
This tone states that the last field keyed in was accepted correctly. You are now positioned at the next field.



C.O.D.

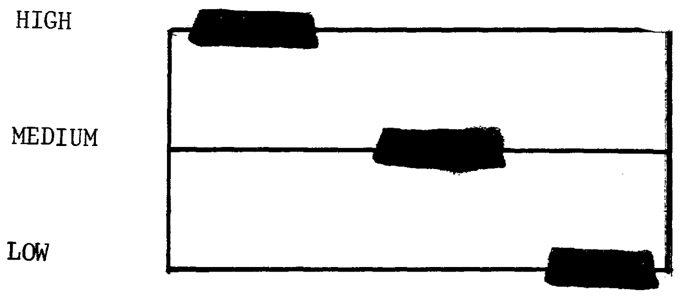
This tone states that the customer who has an open account has now exceeded his credit limit.

This tone is a "Warning Only" and acts the same as a proceed tone. You will receive this tone every time you enter data in the case or bottle quantity fields.



Call in

This tone states that you must call the office after you've completed your order. You will receive this tone after you have keyed in your salesman security code number. This takes the place of a proceed tone.



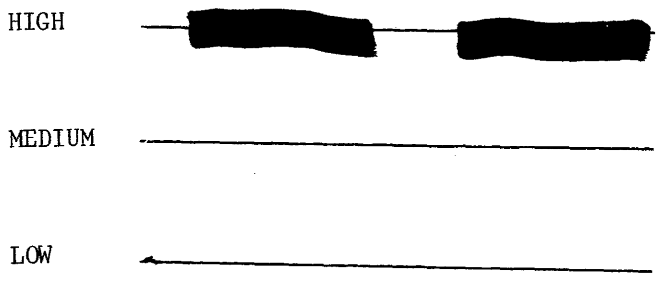
Insufficient quantity

This tone states that we cannot fill your order. You may get this tone either in the case and/or bottle field.

This is an "Action" tone, stating that you must key in one of the three options!

04* PARTIAL ORDER—Give me all that you have and backorder the rest.
06* BACK ORDER—Entire quantity is backordered.
08* CANCEL—Entire line item is cancelled.

After responding to the backorder, be sure to continue to the next field, especially if you want to insert a discount or a discount code.

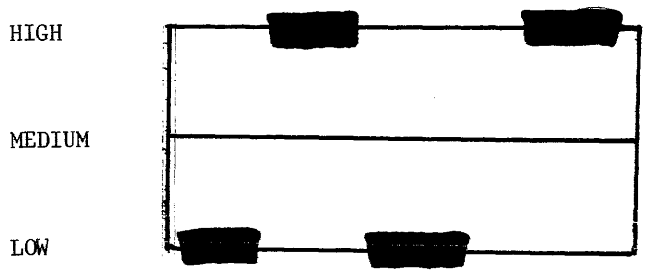


Error tone

This tone states that the last field keyed in was in error. You are now asked to rekey the entry again, correctly. You will be allowed 3 chances to correct your error. If you fail on the 3rd chance, you will be disconnected from the computer.

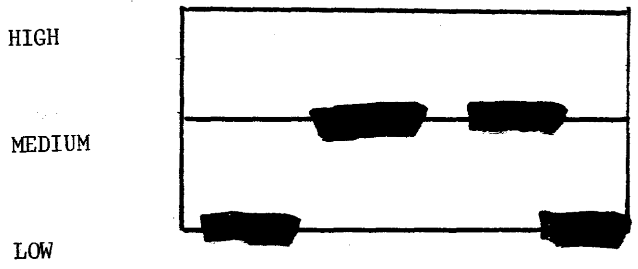
An error may occur during the following fields:

CUSTOMER NUMBER—Bad customer number
PRODUCT NUMBER—Bad product number
DELIVERY OVERRIDE—Invalid delivery code



Rekey

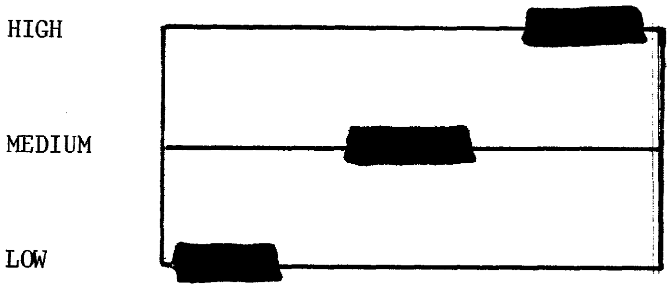
This tone states that you must rekey the last line item, starting back at the product number, due to a combination of errors that has occurred.



Recall

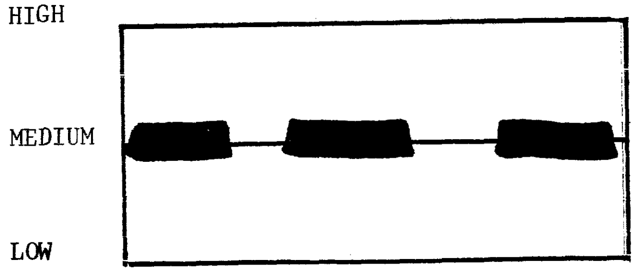
This tone states that you must redial into the computer. The following is a list of reasons why you have been disconnected:

1. You have keyed in the customer number wrong—three times.
2. You have keyed in the product number wrong—three times.
3. The system is down.
4. The customer file is currently in use by another salesman.

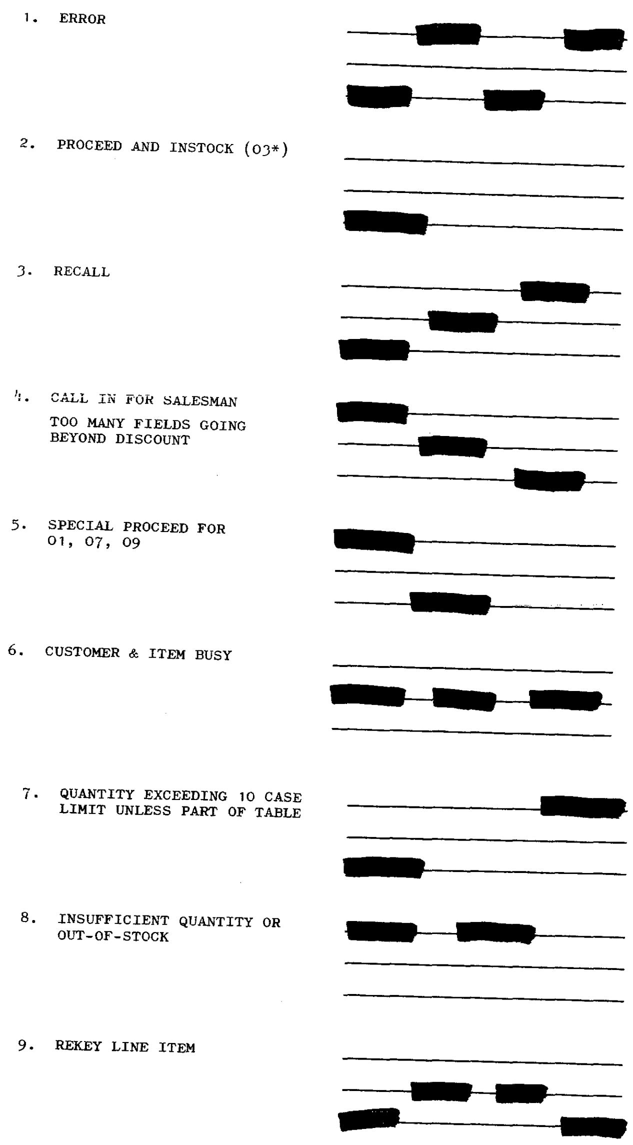


Busy tone

This tone states that the product file is *currently* in use by another salesman. Proceed to another product number and continue your order, coming back to that "Busy" product number later on in that call.



Graphic representation of tones


1. ERROR
2. PROCEED AND INSTOCK (03*)
3. RECALL
4. CALL IN FOR SALESMAN
TOO MANY FIELDS GOING BEYOND DISCOUNT
5. SPECIAL PROCEED FOR 01, 07, 09
6. CUSTOMER & ITEM BUSY
7. QUANTITY EXCEEDING 10 CASE LIMIT UNLESS PART OF TABLE
8. INSUFFICIENT QUANTITY OR OUT-OF-STOCK
9. REKEY LINE ITEM

GUIDELINES FOR OPERATING TOPS

Symbols representations

*—This is used to end a field of data.
(There is a 60 second disconnect timer)
This is also used to skip a field of data.

#—This is used to cancel a field of data.

09*—"End of customer order"
This is used to end a customer order. It will also end a line item (01*) and a customer order at the same time.
Everytime you enter this code (09*), you will be back at the salesman security field. You are then ready for a new customer order.

01*—"End of line item"
This is used to end a line item. Everytime you enter this code, you will be back at the product code field.

08*—"Cancel line"
This is used to cancel an entire line item. In order for it to be effective, you must not have entered an 01* or 09* for that line item.

77*—"End of call"
This should be used when you have completed your last customer order and now you are ready for hanging up the phone.

Where am I? 02*

This question is used for your benefit when you feel you are unsure of what field you are presently in. You may use this at any time, remembering that there are 2 levels of tones, high and low.

The low level beep tells you that you are in Section 1.

1	2	3
Salesman	Customer	Delivery Override

The high level beep tells you that you are in Section 2.

Order

1	2	3	4
Product Number	Case	Bottle	Discount or Discount Code

Deal Discount

1	2	3
Deal Type	Deal Cases	Deal Bottles
4	5	6
Discount Amount	Invoice Number	Deal Number

Deal Order

1	2	3
Product Number	Case	Bottle
4	5	6
Disc. Code or Amt.	Deal Type	Deal Number

Regular Order

FIELD	1. Salesman—Security Number	SECTION 1
FIELD	2. Customer Number	
FIELD	3. Delivery Override	
FIELD	1. Product Number	SECTION 2
FIELD	2. Case	
FIELD	3. Bottles	
FIELD	4. Discount Code OR Discount Amount	

1. Monthly Promotion
2. High Discount
5. Net
6. Entire Order is Net

REGULAR ORDER

Field-#		Tone-#	Reason for Tone	Action to be taken
A.				
1.	Salesman	Proceed	A good salesman security code has been keyed in.	Continue to the next field and enter the customer number.
2.		Recall	System is temporarily down.	Re-dial 398-6100
3.		Call in	There is a message for you to call the office.	Continue normally with your order entry. Then call the office.
4.		Disconnect	There is no tone associated with this. You have entered an invalid salesman security #.	Re-dial 398-6100
B.				
1.	Customer	Proceed	A valid customer # has been entered	Continue to the next field and enter the delivery over-ride number.
2.		Error	An invalid customer # has been entered	Re-key the customer #
3.		Recall	Three bad customer #'s have been keyed in. Customer account # is currently in use by another salesman	Before re-dialing 398-6100, make sure that you have the correct number or call the office for assistance.
4.		Re-key	Too many errors have been entered.	Re-enter the line item beginning at the salesman number.
C.				
1.	Delivery over-ride	Proceed	A valid delivery over-ride instruction has been entered.	Continue to the next field and enter the product #.
2.		Error	A bad delivery override instruction has been entered.	Re-enter with a good delivery instruction.
D.				
1.	Product	Proceed	A valid product # has been entered.	Proceed to the quantity field.
2.		Error	A bad product # has been keyed in.	Re-enter with a valid product number.
3.		Recall	1. Three bad product #'s have been keyed in.	Before re-dialing 398-6100, make sure you have the correct number or call the office for assistance.
4.		Re-key	Too many errors have been encountered.	Re-enter the line item, beginning at the product #.
5.		Busy	Product # is in use by another salesman.	Continue to another product # and then re-try the "busy" product number.

Field-#	Tone-#	Reason for Tone	Action to be taken
E.			
1. Case Quantity	Proceed	Good entry order is ready to be filled.	1. Proceed to the bottle field. 2. Enter an * and then enter discount code or amount. 3. End the line item 01*. 4. End the customer order 09*.
2.	C.O.D.	Good entry order is ready to be filled, but the warning tells you this order is C.O.D.	Same as above (line 1)
3.	Insufficient Quantity	The quantity order can't be filled.	1. Enter 04* Partial Order (fill what you can and back-order the rest). 2. Enter 06* Backorder (the entire quantity is back-ordered). 3. Enter 08* Cancel (the entire line em is cancelled).
F.			
1. Bottle Quantity	SAME AS CASE QUANTITY		
G.			
1. Discount Code or Amount	Proceed	Good entry	1. Enter 01*, ending that line item. 2. Enter 09*, ending the customer order.

Deal Order

Field-#	Tone-#	Reason	Action
A. Salesman		Same as regular order	
B. Customer		Same as regular order	
C. Delivery Override		Same as regular order	
D. Product		Same as regular prder	
E. Case Quantity		Same as regular order	
F. Bottle Quantity		Same as regular order	
G. Discount Code or ammount		Same as regular order	
H. Deal Type	Proceed	Deal type accepted	Proceed to the deal # field
I. Deal type	Proceed	Deal number accepted	1. enter 01* ending that that line item 2. enter 09* ending the customer order

DEAL ORDER

FIELD	1.	Salesman Security No.	
FIELD	2.	Customer No.	SECTION 1
FIELD	3.	Delivery Override	
FIELD	1.	Product No.	
FIELD	2.	Case Quantity	
FIELD	3.	Bottle Quantity	
FIELD	4.	DISCOUNT	SECTION 2
FIELD	5.	Deal Type 8 W - Begin Deal 9 X - Part of Deal	
FIELD	6.	Deal No.	

DEAL DISCOUNT

FIELD	1.	Salesman Security No.	SECTION 1
FIELD	2.	Customer No.	
FIELD	3.	Delivery Override	
FIELD	1.	Deal Type No. Discount Due Z 99770 Completion Y 99880	SECTION 2
FIELD	2.	Deal Cases	
FIELD	3.	Deal Bottles	
FIELD	4.	Unit Discount Factor	
FIELD	5.	Invoice No.	
FIELD	6.	Deal No.	

DEAL DISCOUNT

Field - #		Tone - #	Reason	Action
A.	Salesman			
B.	Customer			
C.	Delivery Override			
D.				
1.	Deal type number	Proceed	A valid deal order # has been entered.	Proceed to the deal quantity field.
2.		Error	A bad deal order # has been entered.	Re-enter with a valid deal order #.
3.		Recall	1. Three bad entries have been keyed in. 2. An invalid deal order number has been entered.	Before re-dialing 398-6100, make sure you have the correct #, or call the office for assistance.
4.		Re-key	Too many errors have been encountered.	Re-enter the line item beginning at the product #.
5.		Busy	Deal order # is in use by another salesman.	Continue to another deal order # and then re-try the "busy" deal order #.
E.				
1.	Deal Cases Quantity	Proceed	Good entry	1. Proceed to the deal bottle quantity field 2. Enter an * and then enter the discount unit amount.
2.		C.O.D.	A good entry, but the warning tells you that the order is C.O.D.	SAME AS ABOVE (line 1)
F.				
1.	Deal Bottle Quantity		SAME AS DEAL CASE QUANTITY	
G.				
1.	Deal Unit Discount Factor	Proceed	Good entry	Proceed to the next field and enter the invoice #.
H.				
1.	Invoice #	Proceed	Good entry	Proceed to the deal # field.
I.				
1.	Deal Number	Proceed	Good entry	1. Enter 01*, ending that line item. 2. Enter 09*, ending the customer order.

8. A busy tone indicates that a customer record or product record is in use. What should you do if this happens?

____

____

____

____

9. List 5 requirements for entering a discount due.

____

____

____

____

10. What happens if you enter no quantity? What is your course of action? ______________________________

____

____

11. Give me 3 reasons why you would receive a warning tone in the ________________________ field?

____

____

12. What happens if you go beyond the discount field?

____

____

____

TOPS QUIZ

1. A ________________ is used to end a field.

2. A ________________ can be used to cancel a field, provided ________________ has not already been entered.

3. If a ________________ has been entered, a ________________ should be used to cancel a line. When this is done, your next field to be entered will be ________________.

4. To end a line, enter a ________________. This will return you to ________________. To end a customer order, enter a ________________. This will return you to ________________.. To end a call, enter ________________ and hang up.

5. An out-of-stock tone will only occur after you have entered ________________, or ________________. You must respond in one of the following ways:

   ____________________, ____________________, or ________________.

6. If you have become distracted and don't know where you are, enter ________________ and the computer response will lead you to the proper field.

7. If the computer responds with an error tone, what course of action is required?

   ____

   ____

   ____

   ____

Use of a multi-programming mini-computer for real time control applications

by FREDRIC C. JAYE
Environmental Protection Agency
Research Triangle Park, North Carolina

and

RONALD J. REINER
Systems Technology Associates, Inc.
Falls Church, Virginia

INTRODUCTION

During the spring of 1973, the Environmental Protection Agency Research Center at the Research Triangle Park, North Carolina, began the process of constructing a medium scale wind tunnel for research in stationary source air pollutant emissions measurement. This stationary source simulation facility consists of a closed loop stainless steel wind tunnel capable of 20 meters/second velocity and 200°C operation with a wide variety of gases, particulate matter, and other contaminates circulating in the gas stream. The combustion products CO_2, CO, and H_2O are generated by a million BTU/hour combustion unit. The remaining H_2O is injected directly as steam as are the sulfur dioxide and nitrogen oxides. The particulate matter is suspended in a large fluidized bed aerosol generator and aspirated into the wind tunnel.

Since manpower to operate this facility was limited, we have attempted to automate the controls as fully as possible.

With the introduction of the 960A computer in late 1971, Texas Instruments Corporation introduced a relatively low-cost multi-programming operating system called Process Automation Monitor (PAM). Similar capabilities come from Digital Equipment Company in RSX-11, Data General in RTOS and Modcomp in Max II or III.

Prior to the availability of this type of operating system for these 16 bit mini-computers, we could have handled the multi-task control system for this source simulator in one of three approaches:

The traditional mini-computer technique would be one massive program with a number of subroutines at many levels to accomplish the various tasks. This program would be written in assembly language and offer the maximum efficiency in terms of memory utilization but at some cost in program preparation time and at considerable cost in agony units required for debug and later modification.

Another approach to the problem would have been to use an existing multi-programming system, such as MPX on an IBM 1800 or RBM (Real Time Batch Monitor) on a XDS Sigma 2.

The third approach would be to write our own mini-monitor system to handle the essential functions of the multi-programming system.

The availability of relatively competent multi-programming operating systems for the new group of mini-computers has made control applications programming substantially easier for time flexible applications. By time flexible, we mean that the external event timing and required response were an order of magnitude, or more, longer than the operating system response times for various functions.

The utilization of the approach we will describe is always subject to timing constraints. However, in most real world applications, especially those involving large physical systems, the system overhead incurred is tolerable, and the ease of programming and operation more than outweigh the system overhead penalties.

The tasks which the source simulator data and control system is required to do are fourfold.

First, Fault Checking and Alarm Indication. Built into the various simulator components are some 31 fault indicators, status warning lines, or hazzard sensing units. Typical of these are pressure or temperature switches to sense over or under limit conditions, analog output combustible mixture alarms, and area carbon monoxide monitoring units. The Data and Control System is employed to check periodically the status of these devices, perform alarm functions, print operator information messages, and in some cases, take specific automatic corrective action.

The second task is the supervision and control of some 10 channels of permanently assigned instrumentation on the simulator. This includes CO, CO_2, H_2O, SO_2, NO_x, O_2, area CO, combustible mixture alarm, temperature, and velocity sensors. At the beginning and end of a test (as well as when required during a test), the operator may request the Data and Control System to zero and calibrate any of these permanently assigned sensors.

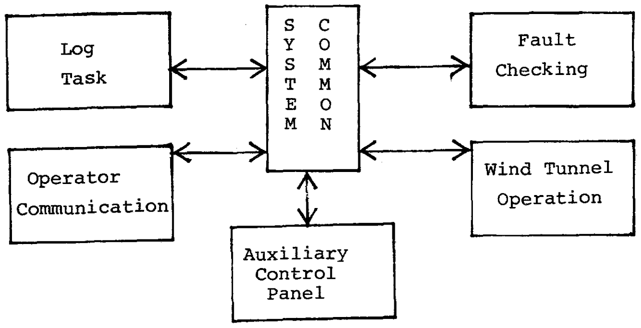


Figure 1—System organization

The third task involves the usual job of data acquisition during the test program. Here the system samples up to 27 channels per test on selected time cycles, digitizes, converts to engineering units, and performs the usual storage and retrieval tasks.

The fourth major function is that of integrated closed loop control of the temperature, velocity, water, sulfur dioxide, nitrogen oxides, and particulate content of the source simulator gas stream. Before a test is actually run, the operator inputs a table of values for each of these parameters and the times at which these values are to be implemented. During the test, the software system cycles through this table, updating the actual parameters whenever necessary.

SYSTEM ORGANIZATION

The Texas Instruments Model 960A is a 16 bit mini-computer using 900 n sec. semiconductor memory. We have the maximum internal memory of 32,000 words. The unit is capable of external expansion to 65,000 words. The internal architecture uses two sets of registers, one for supervisor mode and one for worker mode. Each set contains 8 memory addressable registers.

All the registers are general purpose and may be used for indexing or for computation, however, registers 4, 5, 6 and 7 of each set supply additional services.

Registers 4 and 5 are useful for designing reentrant programs. Register 4 is hardware implemented as a default index register for several data reference instructions. Hence, it can act as base register for all data within a task. Register 5 is a default index register for some branch instructions. So it can be used to index the pure procedure portion of a task.

Register 6 serves as a base register for 'bit' reference instructions. Register 7 is hardware implemented to act as a base register for I/O processing.

If the conventions concerning these special purpose registers are adhered to within a task, the task will be fully relocatable. This arrangement also provides for full reentrancy, since a pure procedure will operate on its data via registers 4, 6 and 7.

The heart of the I/O processing structure in the T.I. 960 is the Communications Register Unit (CRU). All I/O devices, except for those connected through the DMAC, terminate in CRU lines which are individually addressable with register 7 acting as a bias register. Functionally, the CRU can be thought of as a large I/O register (maximum 8192 bits) which can be addressed on a bit-by-bit basis, on a variable field basis or on a 16 bit word basis. This structure allows a great deal of flexibility in terms of peripheral interfacing. It also simplifies the necessary software drivers. An example of CRU processing is given in Figure 3.

All task I/O and other program-operating system interactions are done by means of supervisor calls via register 3 and switching to supervisor mode. Thus, until a worker program ceases to execute (suspension, I/O wait, etc.) the program counter and other registers are not disturbed to process operating system calls. The supervisory structure provides the facilities for a task to suspend itself for a given time period or for one task to unsuspend another.

The 960A supports two multi-programming operating systems—PAM Process Automation Monitor—an all memory resident system and PAM/D PAM (DISC) which supports disc or memory resident tasks, data files and program overlays. The system is capable of supporting 256 tasks and 256 logical units. All I/O is via logical unit definitions unless directly to the CRU.

Texas Instruments has recognized the need for high level language programming in process control. To this end, they have provided a FORTRAN superset called Process Control Language (PCL) which facilitates CRU and 'bit flag' processing by means of declarations within a PCL program. Given these capabilities, any process control application can be written in this extended FORTRAN. We used PCL for the majority of our programming effort, i.e., wherever space or time considerations were not critical.

PROGRAM ORGANIZATION

In the introductory section, we discussed the four major tasks involved in the operation of the source simulator. We have taken advantage of the multi-programming aspect of the PAM/D monitor to segment the programming into seven major tasks, all multi-programmed under the standard operating system.

The Data/Control System programs are:

Task 21	TDFLC—Time Delay Fault Check
Task 24	Auxiliary Control Service
Task 26	System Common (SYSCOM)
Task 27	Fault Check Message Writer
Task 28	Wind Tunnel Driver (WT Driver)
Task 29	Operator Communication (OPCOM)
Task 35	Sort/Log
Task 55	Start Up/Shut Down

Task 26—System Common

SYSCOM is the task responsible for data communication between all the other tasks. The design considerations that went into implementing SYSCOM are very important since SYSCOM represents the integration effort for the entire system. The design chosen was quite effective. After the other six tasks were written and debugged independently, the actual software integration process at the STA facility took less than two weeks.

Some of the features which were considered in SYSCOM were:

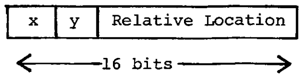
1. It should be secure from array overruns affecting other data elements in the interface.
2. It should be flexible, so that whenever data structures, change or are added, no major system revision is required.
3. Since the data flow between tasks represents the state of the overall system, the data itself should be centralized to facilitate debugging during integration.
4. SYSCOM should be available as a dummy interface while individual modules are being debugged and implemented. As it turns out, our system communication structure was a valuable debugging aid in itself.

In its final form, SYSCOM is a totally passive task which acts as a data repository and communication area. As such, it meets requirements 3 and 4 above. Its internal structure, which is the topic of the following paragraphs, indicates its conformance to the other two requirements.

SYSCOM contains a small procedure portion, which serves to unconditionally suspend the task when it is loaded. The data portion of the task has two components, a pointer area and actual data storage.

All shared data elements in the system are assigned a reference pointer location in the pointer area. The pointers are all unique, and the index of the pointer in the table provides a consistent means of referencing the data.

These pointers are structured as:



```
where x is o = scalar data
           1 = array data
       y = 0 = integer data
           1 = real data
```

'Relative Location' is the location of the true data relative to the first word of the task. The data structure for an array uses data word one as a true array length indicator, thus preventing possible overflows.

By referencing data in this manner, we can easily change data array lengths, or constants, etc., without drastic re-compilation of all the other tasks. Each operating task contains only the data which that task and its subroutines actually uses.

Communication with SYSCOM by any operational task is handled by a relocatable FORTRAN subroutine 'CDAT'. This routine is entered by CALL CDAT (ICOM, ITASK, LENG) where:

ICOM is the system common pointer location.

ITASK is a FORTRAN common location within the calling task.

LENG represents the number of words to transfer. Its sign indicates direction of transfer. If LENG is positive, data will be transferred from SYSTEM COMMON to FORTRAN COMMON of the task. If LENG is negative, data will be transferred to SYSTEM COMMON. Upon return, LENG contains the actual number of words transferred.

This subroutine uses a supervisor call which returns the first word address of each task as actually resident in memory. This address is used to compute the actual address of the data pointer in the system common task.

Several of the key data structures in the task are:

DATA WORD 1—1 word integer—system mode
= 1—initialize
= 2—run
= 4—pause
= 8—reset elapsed time
= 16—reset all data tables

DATA WORD 29—78 word integer array

13 time value	0-xxx minutes
13 velocity values	0-xxx FPS
13 temperature values	0-xxx Degrees F
13 sulfur dioxide values	0-xxx PPM
13 nitrogen oxides values	0-xxx PPM

DATA WORD 26—8 word integer array
year of last log cycle
month of last log cycle
day of last log cycle
hour of last log cycle
minute of last log cycle
second of last log cycle
mode = 1 normal log rate
= 2 fast log rate

DATA WORD 32—6 word integer array (single) permanent data channel information
word 1—A/D channel (0-40)
2—A/D channel gain
3—full scale value engineering units
4—zero value
5—allowable zero deviation
6—allowable calibration deviation

DATA WORD 61—3 word temporary data channel information
1 A/D channel #
2 A/D channel gain
3 Full scale value engineering units

```
CRU addresses are hardware biased by register 7.

Example:

     Register 7 contains ' D '          (CRU card slot 13)

        SETB  2, 1             Sets Bit #2 of card 13
        BBNE  9, 0, not        Branch to sumbol "not" if Bit 9 is set
        LDCR  (4,8),value      Load an 8 Bit wide field within
                               the CRU starting at Bit 4 from
                               location "value"
        STCR  (32,3),value     Read Bits 0-2 of Card F into
                               location "value"

     Field widths and bit locations may be assigned FORTRAN
     variable or assembler symbolic names.
```

Figure 2—CRU addressing

DATA WORD 76—2 word array integer
1 test file logical unit #
2 # of sectors in test data file

A major problem encountered in this design stems from the sharing of the same data by two or more independent processes. If one process can change the data at any time, another process can never by assured of data integrity.

The overall system design obviated any need for concern in most cases. The cases where it was of concern occurred only whenever one task set a command flag for another task. For example, reference pointer 6 of System Common is used by OPCOM to start and stop the SORT/LOG task. And SORT/LOG uses it to inform OPCOM that it has completed a specified function. Judicious design is all that is necessary to avoid any timing conflicts in this case.

Task 21

Time delay fault check is used to provide different "bid" cycles to the true fault check task. (27)

Task 24

Auxiliary control panel service has been added since the system was put into operation and is based on a good deal of operating experience. The principal commands given via OPCOM once a test is begun are the mode commands, "run, pause, continue" or changes in the data log rate "fast or normal." Therefore, we built a small hardwired auxiliary control panel with a series of push buttons to signal various functions. All these functions are wired into a 16 in/out CRU board at TTL levels. A series of OR gates will generate an interrupt on the same CRU module when any of these buttons is depressed.

The coding for this task is shown in Figure 3. We feel that it illustrates several important aspects of our system and the programming approach we have taken.

First, the program is implemented largely in PCL the high level FORTRAN for this system. Second, implementation of this processor required 2 man-days to write, install, and debug. Third, it added an important capability to the system with minimal changes in the other operating tasks. In fact, the only software change made was to Task 29, OPCOM, to check a contact closure at the end of its loop. If the contact is closed, the task loops and waits for further TTY input, if not the task terminates via an "end of program" supervisor call. This facility allows OPCOM to reside on the disc when not needed.

The important point illustrated here, however, is the modularity of the programming under a multi-programming system and the resulting ease of addition of major processors or system modifications.

It also illustrates the combination of high level and in-line assembly programming which is available in most minicomputer systems today.

Task 27—Fault Check Message Writer

As originally written, this task is loaded into memory and suspended. At 10 second intervals, it is unsuspended and bid via a supervisor call from Task 21. The task surveys some 18 contact closure or sensing lines which are triggered by various external devices. This task also samples 3 A/D

```
      COMMON ISTATUS, ILOGG
      DIMENSION ILOGG (8)
      DATA LON/1/, LOFF/0/, LRUN/2/, LPAUSE1/4/, LENG/8/, LOG/26/
 Z1   LA 7, X'0F0'           Set CRU Base Register
      SETB 14,0              Enable Interrupt on Line 14
      LA 3, X'24FE'
      SBSX * SENTRY          Suspend Task Waiting for Interrupt
      BBNE 1, 0, Z10                  Initialize Mode
      BBNE 2, 0, Z20                  Run Mode
      BBNE 3, 0, Z30                  Pause
      BBNE 4, 0, Z20                  Continue
      BBNE 5, 0, Z50                  BID OPCOM
      BBNE 6, 0, Z60                  Start Wind Tunnel
      BBNE 7, 0, Z70                  Fast Log
      BBNE 8, 0, Z80                  Normal Log
 10   ISTATUS = 1
 15   CALL CDAT (LONG, LON, LON)
      GO TO 105
 20   ISTATUS = 2
      GO TO 15
 30   ISTATUS = 4
      GO TO 15
Z50   LA 3, X'0829'                   BID TASK 29 OPCOM
      SBSX * SENTRY
      GO TO 105
Z60   LA 3, X'0828'                   BID TASK 28 WIND TUNNEL DRIVER
      SBSX * SENTRY
      GO TO 105
 70   CALL CDAT (26, 2, 8)            GET LOG STATUS WORDS FROM COMMON
      ILOGG (8) = 2
 75   CALL CDAT (-26, 2, 8)           WRITE NEW LOG STATUS
      GO TO 105
 80   CALL CDAT (26, 2, 8)
      ILOGG (8) = 1
```

Figure 3—Partial Fortran and assembly coding for auxiliary control panel

channels and compares the values with preset limits. An appropriate message is output to the operator if various fault, hazard or warning conditions exist.

During normal operation, this task is disc resident waiting an interrupt call from Task 21.

During start up and combustion unit operation, the task is memory resident and scanning at 5 second intervals. The advisory type messages may be overriden via OPCOM. The hazard warning messages require immediate action and are repeated until the condition is cleared. This task uses the message writer spool/unspooling facility available in the supervisor.

Task 28—Wind Tunnel Driver

This task is the heart of the control system for the source simulator. It is memory resident after the SWDT (start wind tunnel) command is given via Task 24 or OPCOM. It contains several major sections all operating serially in a basic 5 second time delay loop. The task is structured into FORTRAN subroutines for A/D reading, engineering unit conversion, temperature, velocity, humidity, sulfur dioxide, and nitrogen oxides control and velocity display.

The A/D readings, conversion and velocity display are updated every 5 seconds. The control subroutines normally make corrections every 3rd pass or each 15 seconds.

Task 29—Operator Communication (OPCOM)

This task is the most complex of the system tasks. The main structure of the task consists of several subroutines to accept the command input from the console typewriter. These commands then enter a comparison and skip chain to identify the command.

System mode commands (run, initialize, pause, continue, fast, norm) all change various flags in the system common task. The more complex commands all load various disc resident overlays to perform the more complex processing. These commands set up the different parameter tables, select instrumentation, perform zero and calibration functions and override fault messages.

The structure of OPCOM is shown in Figure 4. As you can see, we have a two level overlay scheme for some of the set up commands. We will not discuss the command function further except to comment that the non-overlaid length of the program was 22,000 words and the overlaid length is about 4800 words most of which is overlay space.

Task 35 Sort/Log

This task performs the two major jobs implied in the task title.

Upon initiation of the "run" command via OPCOM, the Sort/Log tasks writes a data header on the first 13 sectors of the disc test file. This test file is usually 120 sectors long

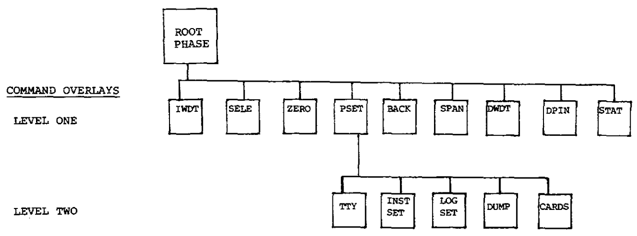


Figure 4—Operator communications task structure

and is assigned by logical unit number. The complete instrumentation and setup parameter tables are copied from system common. This has two major advantages: first, a test file may be saved for later sorting and data analysis with all definitions intact and second, a test may be suspended and restarted with the same parameter information. Handling this header information is the function of the two overlays "OOTF" "open old test file" (and restore definitions into system common) and "ONTF" open new test file and copy definitions from system common.

The root phase of Sort/Log acts as a dispatcher and time counter. Data logging intervals are obtained from system common and compared to the system clock as a day, hour, minute, second elapsed time since last log basis.

The log overlay calls for an A/D conversion of table selected channels and writes the binary result to the disc. Up to 26 channels may be defined per test which results in a 32 binary words second which is one sector. The remaining 6 words are used for time information.

As each sector (a record) is written, the next sector is set equal to -1 (HEX-'FFFF"). This is a moving "EOF" signal to the sorting program so that a test may be interrupted at any time for almost any reason which does not destroy the data on the disc and all the previously logged data will be retrievable.

After the test is terminated, the sort overlay of this task is used to ask for data channels by name prior to sorting. We can sort up to five channels at a time. There is no limit to the number of times a channel may be sorted. Thus, the data can be sorted and printed by channel vs. time. The conversion to engineering units is handled during the sorting period. Since the output is by logical unit, it is convenient to assign temporary disc files to the output data and, thus, have it readily formatted for input to regression or other data analysis program. For example, for a multiple linear regression program, we specify that the dependent variable be column 1 followed by up to 4 independent variables. The analysis program in this case is set to ignore the headers and time information.

This concludes our exposition on the software construction of our source simulator data and control system. Now we are ready to discuss that long and painful task of actually bringing the system up and interfacing with the physical world.

The system was delivered from Systems Technology Associates to the EPA Research Center in North Carolina on 15 August. Hardware reassembly, program loading and checkout was completed on the 17th of August.

Connections between the computer system and the source simulator control panels were completed on Monday and Tuesday of the following week. Start-up and operation of the facility under full computer control and supervision was accomplished on Wednesday afternoon. To date, the only adjustments and changes to the programs have been in the area of patching various control constants in the WT DRIVER task and placing the fault check task under interrupt control.

We attribute this extremely fortuitous situation to several major factors. First, the multi-programming operating system made modular programming not only possible but very desirable. This allowed task by task checkout before requiring full system operation. Second, we made extensive use of an electronic break-board simulator to verify contact closures, relay outputs, A/D inputs and D/A outputs. Third, the task of interfacing to the external world was made immeasurably easier by the wide variety of CRU modules available from Texas Instruments and a CPU which was designed for this type of work. Fourth, an early decision was made to put all input/output line references and control and data constants into FORTRAN "data" statements in each program, instead of in-line coding. This probably cost us some memory and efficiency in terms of memory reference instructions and large symbol tables but it made it possible to patch all the control constant changes usually required rather than recompiling, relinking and reloading the programs. For example, the velocity controller required 7 volts from the D/A connect to deliver full power instead of the 6 volts we expected. The repair was accomplished by patching one integer constant and one "mask" in two subroutines. Fifth, the interfacing was assisted by a rigid adherence to a set of defined interface rules which were worked out between the computer system team and the control system designers.

In conclusion, we have discussed how we used a typical multi-programming operating system as is available on a number of the mini-computer systems currently in use. Our particular on-line control of five variables in a specialized wind tunnel is admittedly not a complex nor a particularly time critical task, however, we have not stressed the capabilities of the system in any way. Our particular system happens to have a primitive sort of duty cycle calculator built in the operating system which calculates the duty cycle for the last second and stores the current and maximum value experienced since system initialization. The maximum duty has been 42 percent experienced one day while running a full scale source simulator test and assembling another program in the background. The normal duty cycle while controlling the simulator is about 26 percent.

Thus, on the basis of our experience, we would recommend that in control or data acquisition situations where microsecond response to a stimulus is not required, that the design teams consider carefully the programming and operating advantages offered by the current mini-computers and their multi-programming operating systems.

A data bank for on-line process control—The synchrotron injector

by A. DANEELS

CERN
Geneva, Switzerland

INTRODUCTION

The need for a data bank results from the complete computerization of the CERN Proton Synchrotron injector process. This process was designed so as to allow on-line control of most of its variables and a complex software system was developed for multiple and simultaneous control. Emphasis was put on "interactive control" i.e. several operators may control in parallel part of the synchrotron injector through computer-driven consoles. Conflicting control has to be avoided and therefore a centralized information base or data bank was developed.

Broadly speaking this data bank is subdivided into two complementary parts: a very small one, which is permanently in core, and a large one (at present around 10 k), which is stored on disk.

The small core-resident section contains information which is likely to undergo frequent changes: a few "flags" called "cold start" flag (to indicate which part of the process is under no control after a cold start of the computer); "BUSY flag" to indicate which part of the process is under control from the MAXI-CONSOLE; "MANUAL flag" to indicate which part of the process is under control from its dedicated MIDI-CONSOLE; the starting address of a binary matrix containing all "status" information, such as "ready" or "not ready," "on" or "off" and "polarity" status of the process variables. This matrix is built up of a number of 16-bit words (so far 25 words) in which every single bit represents a status (ready/not ready, or on/off, or polarity). Random arrangement of status bits within the matrix eases the hardware aspect of collecting this status information from the various variables. This matrix is permanently in core as part of a surveillance program, which is imbedded within the MIDI-CONSOLE[1] program and which scans the various status bits at regular time intervals.

Nothing more needs to be said about this core resident section of the data bank. This paper gives a detailed discussion of the characteristics of the disk-based section of the data bank.

GENERAL

The control of a process is in general implemented by a great many control variables of different types.

To vary any control variable one needs to have some information at hand, as to its current control value, whether this variable is controlled via, for example, a stepping motor, what is its possible maximum value, whether it is switched on or off, etc. It will be shown here how, in general, one needs around twenty such pieces of information which are called the characteristics of every control variable.

"Control variable" has been mentioned so far without indicating whether they are single control elements, or several control elements combined into one control vector. Throughout this paper distinction will be made (where needed) between *control elements* and *control vectors*, where the latter are defined as a linear combination:

$$\mathbf{X}=\sum_{i=1}^{n}\alpha_i x_i. \tag{1}$$

where

$\mathbf{X}$ is the control value of a vector,
x_i is the control value of the ith element,
α_i is a weighting factor,
n is the number of elements.

Any statement about control variables is valid for either elements or vectors.

Any program controlling all or part of the process (e.g., on-line optimization) needs to know the characteristics of its control variables, which therefore should be stored, for example, on magnetic disk.

A process such as the synchrotron injector has around 500 variables each characterized by 20 pieces of information, hence one will need space for around 10,000 pieces of information. This plunges us into the problem of data banks, data structure, information retrieval and updating.

It will be shown how a data bank for control variables may be given a simple structure.[2]

It will also be shown how to define every control variable by a single code, SOFT-CODE, for retrieving its characteristics.

Eventually a software system will be described, which allows easy retrieving and updating some of the characteristics without the need for direct access to the data bank.

CHARACTERISTICS OF A CONTROL ELEMENT

A control variable has been defined (see the previous paragraph) as either a control element or a control vector. This section will describe the characteristics of every single control element. The control vectors will be dealt with in a subsequent paragraph.

Control element's name

This is for the purpose of displaying the name of the control element on alphanumeric displays as, for example, on the MIDI-CONSOLE.[1] Such a name is in general a mnemonic indicating to which group the control element belongs, for example, a vertical steering magnet in the injection line of the synchrotron injector, I-DV, followed by an item number, e.g., the 7th one. This full name is thus

I-DV07.

Consequently a full name will be a mnemonic of four alphanumeric characters indicating the group followed by a 2-digit item number. This arrangement covers a large number of possible "names."

Physical units

Once again for displaying the measured value of a control variable one prefers this value to be indicated in physical units: for example, a current, even though measured as a voltage over a shunt, should be displayed in A or mA; an angular position, even though measured, for instance, as a voltage, should be indicated in mrad, etc. In general two characters will be sufficient for defining the physical units.

A BUSY indicator

Any control variable may be controlled from various places. This is particularly true in a multiprogramming environment. To avoid conflicts one needs a centralized indication to tell whether a control variable is already under computer control or not. This indication is available in a "flag" as part of the core-resident section of the data bank (cf. Introduction).

Value of the least significant bit (LSB) at acquisition

Suppose one measures a current. In general this is done through a shunt resistor with a fixed value so that one actually measures a voltage which depends "linearly" (assume the shunt is linear) on the value of the current. The ratio voltage/current depends on the value of the shunt resistor. The analogue value of the voltage is now converted into a digital value, so that going back through all the various conversions one finds a definition of the LSB (i.e., digital value = 1) as a function of the physical unit, e.g.,

LSB = 44.6 mA.

Position of the decimal point

In previous examples, the smallest value one could measure corresponded to the value of the LSB (digital value = 1) at acquisition. Therefore, the least significant digit in physical units is 10 mA or 0.01 A, and consequently if the measured value were to be displayed as a 5-digit number in A one would have to insert a decimal point in front of the second digit right justified: e.g.,

170.15 A.

Value of the LSB at control

Having observed the current value of the control variable, for example, on the display, one decides now to increase or to decrease this value. Therefore, one introduces a positive or negative increment defined in its physical units. This increment therefore has to be processed by the computer and converted into the appropriate digital value before sending it to the digital control register of the control variable: this implies that one should define the physical value of the LSB at control (digital control value = 1). If, for example, the maximum value of a power supply is 15 A and it is controlled by a 10-bit digital register then, of course, the LSB is 14.6 mA.

Maximum and minimum value

This has been mentioned implicitly in the previous section. One needs to be safeguarded against exceeding limits which may not be symmetrical (e.g., unipolar supplies).

Type of control element

One distinguishes, in general, between three types of control elements.

(a) *A-type*

This is defined as the control element whose control value results from the algebraic sum of its previous control value and an increment, for example

$$x_t = x_{t-1} + \Delta_t, \qquad (2)$$

where t is the time of control.

(b) *S-type*

This is defined as the control element whose control value results from an increment only (e.g. *S*tepping motors)

$$x_t = \Delta_t \qquad (x_{t-1} \text{ always} = 0). \tag{3}$$

(c) *C-type*

This is in fact an A-type control variable for which there is no acquisition (measurement) foreseen. This applies more specifically to the preset counters of the synchrotron injector: in particular their control value includes the selection of a proper timing clock.

Indication for READY-NOT READY

Some control variables may be READY or NOT READY depending, for example on whether there is a power supply connected to it or not. This indication is supplied to the computer for most elements of the synchrotron injector.

Indication ON-OFF

This indication allows one to find out whether an element has been switched on or off. If it is found to be switched off the computer should, of course, switch it on before control can start.

Indication POLARITY

For some control elements the polarity of the measurement may be supplied in a different piece of information.

RESET

When an element is found to be NOT READY the computer may attempt to RESET this element.

Acquisition and control addresses

The process computer is connected to the process by a transmission system* through which measurements are acquired and control values sent out. In order to "multiplex" the transmission to or from the proper control elements, every element is given two addresses, one for acquisition, one for control.

Current control value

In order to avoid non-linearities between control value and measured value and also to avoid errors in the acquisition (e.g., missing bits) one may decide not to compute the new control value as the sum of the previous *measured* value and an increment but as the sum of the previous *control* value and an increment. This implies that the computer has memorized what value is actually on the digital control register of any control element.

"Reference" value

This is a control value which was found to be satisfactory in the past as it ensured a good working condition of the process. If the current control value proves to harm the working conditions of the process one wants to return to the "reference" values.

"Buffer"

This allows one to compare the working conditions of the process resulting from the "current control value," with the conditions resulting from the "reference value," without destroying the "current control value" which is stored in the "buffer."

So far the characteristics of a control element have been enumerated. However, in general, some of these characteristics will be common to various control elements. As an example, all vertical steering magnets of the injection line of the synchrotron injector upstream to the distributor are called I-DV, their value is displayed in A, and they are of the A-type, etc.

However, the 7th member of this group will be called I-DV07, and has an individual address for acquisition and control, etc.

So in general one can state that the first eight characteristics will be common to a *group* of control elements, whereas characteristics nine through sixteen will be specific to each element of this group. Hence we come to a two-level data structure with *group characteristics* and *individual characteristics*.

$$\begin{tikzcd} & \text{GROUP(J)} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{INDIV(J,1)} & \text{INDIV(J,2)} \ldots & \text{INDIV(J,M)} \end{tikzcd}$$

CHARACTERISTICS OF CONTROL VECTORS

The control vectors have been defined previously (see the Introduction). A control vector is characterized by:

(i) *the number of its control elements:* e.g., *N*,
(ii) *N codes:* each defining a control element,
(iii) *N coefficients* (which need not necessarily be integers): to weight the control value of every one of its control elements.

DATA BANK STRUCTURE ON DISK

Control elements

The simplest method is to store contiguously the record representing the characteristics of group J, then the char-

* The synchrotron injector uses the digital STAR system (Système de Transmission Adressé Rapide) developed by the Control Group of the CERN Proton Synchrotron Division.

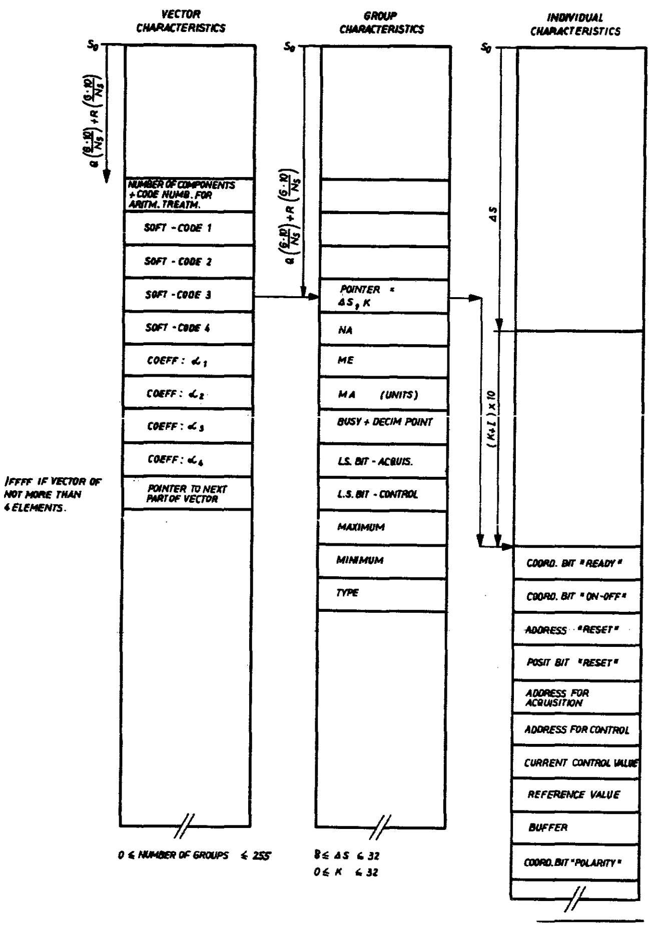


Figure 1

acteristics of all its M individuals, then the specification of group J+1, etc. However the number of individuals M may be different from one group to another and retrieving the characteristics of one single control variable would require, in general, knowledge of two pieces of information: the address (on disk) of its group characteristics and its item number within this group.

This arrangement is also rigid and would not allow, for example, giving different group characteristics to the same string of control variables (e.g., give it another name when various control elements of one group are combined into a control vector).

Consequently, one is led to a more flexible arrangement by storing into two different files the group characteristics and the characteristics of their individuals

...GROUP(J), GROUP(J+1), ...

INDIV(J,3).... INDIV(J,M); INDIV(J+1,1), ...

Each record of group characteristics includes a pointer to the address on disk of the individual characteristics of the first element of this group.

By arranging the record of group characteristics to be of fixed length (10 words) one is able to code both the address on disk of the group record of a control element and its item number into one piece of information. An example of such a coding will be given in a subsequent section and will be called SOFT-CODE (Software Code).

Record length for group characteristics

Characteristics one to eight inclusive have been defined as group characteristics. The "name" has been defined as four alphanumeric characters indicating the group followed by an item number within this group. This item number may be introduced in the code, called SOFT-CODE, which will allow retrieving the characteristics of the control variable. Therefore, it need not be stored on disk. Consequently, one will need in general two words of storage for the name of the group (alphanumeric characters are coded two characters per 16-bit word).

Furthermore, the record of group characteristics will include a pointer to the record of its first element (i.e., item 0).

By appropriate coding of other information such as decimal point position, busy indicator, and value of LSB, one can keep the length of the group record down to ten words (cf. Figure 1).

Record length for individual characteristics

Characteristics nine to sixteen, inclusive, have been defined as individual specifications. The total length of this record is effectively of 10 words, as is shown in Figure 1.

Control vectors

Control vector characteristics are imbedded within group characteristics of control elements. One distinguishes a vector from an element by its SOFT-CODE (see following paragraph).

To preserve the 10-word modular structure a vector of N elements is subdivided into subsets of four elements. The first word contains the number of elements; then four codes, SOFT-CODES, define the elements followed by their corresponding weighting factors α. The last word points to the next following subset and is equal to -1 (hexadecimal/FFFF) in the last subset. However, so far there are no such examples among the synchrotron injector control vectors.

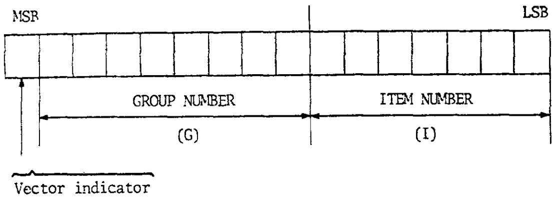
Estimate of the number of sectors on disk

In the particular example of the synchrotron injector, an IBM 1800 process controller is used, with moving search head magnetic disks for mass storage. Every disk has 512 K of 16-bit words storage capacity, subdivided into 200 cylinders of eight sectors each. Every sector has a capacity of 320 words. Hence 32 group characteristic records may be stored per sector. The number of groups of control variables for the synchrotron injector is estimated around 256, which corresponds to eight sectors. Another 24 sectors store the individual characteristics.

SOFT-CODE

Every control variable is defined by a SOFT-CODE retrieving the record of its characteristics in the data bank.

The IBM 1800 process controller which is used by the synchrotron injector is a 16-bit machine; consequently the SOFT-CODE is a 16-bit word which is organized as follows



(i) *Most significant bit, sign bit (MSB)*

This bit is set to 1 for control vectors and is set to 0 for control elements.

(ii) *Group number (G)*

In the case of the synchrotron injector 256 groups have been estimated; hence eight bits.

(iii) *Item number (I)*

The remaining seven bits indicate the item number of a control element within its group.

This SOFT-CODE is effectively a pointer to the address on disk where the characteristics of its corresponding control variable are recorded. This address on disk is computed as follows:

—address of the record of group characteristics =

$$S_o + Q\left\{\frac{G\cdot 10}{N_S}\right\} + R\left\{\frac{G\cdot 10}{N_S}\right\}, \tag{4}$$

where

S_o is starting address on disk of the data bank,
G is the group number,
Q is the quotient of the division between braces,
R is the remainder,
N_S is the number of words per sector (=320 in the case of the IBM 1800);

—address of the record of individual characteristics =

$$S_o + P + (I\cdot 10). \tag{5}$$

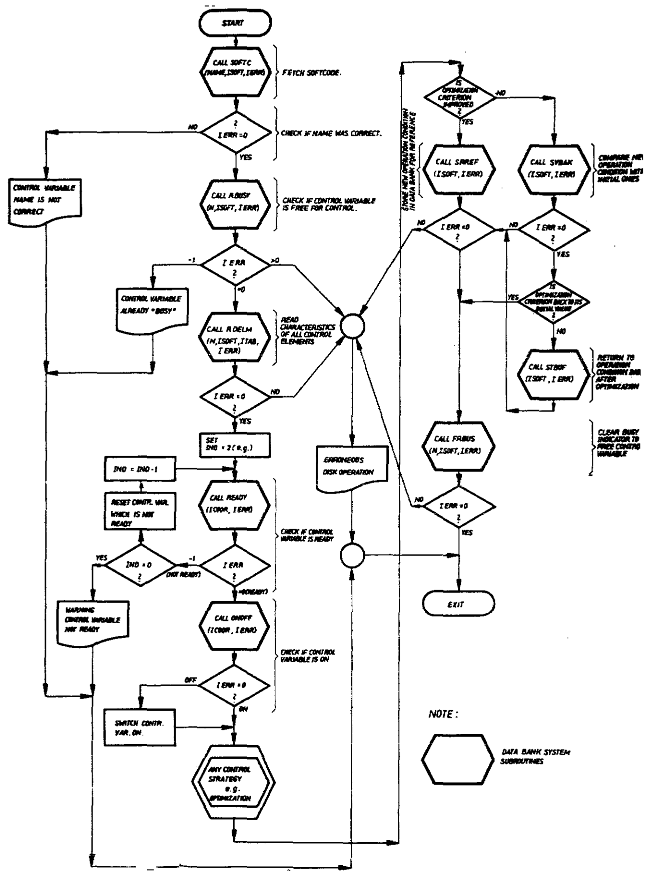

START
CALL SOFTC (NAME,ISOFT,IERR)
FETCH SOFTCODE.
? IERR =0
NO
YES
CHECK IF NAME WAS CORRECT.
CONTROL VARIABLE NAME IS NOT CORRECT
CALL RBUSY (N,ISOFT,IERR)
CHECK IF CONTROL VARIABLE IS FREE FOR CONTROL.
IERR ?
-1
>0
=0
CONTROL VARIABLE ALREADY "BUSY"
CALL RDELM (N,ISOFT,ITAB, IERR)
READ CHARACTERISTICS OF ALL CONTROL ELEMENTS
IERR =0 ?
SET IND = 2 (e.g.)
IND = IND-1
RESET CONTR. VAR. WHICH IS NOT READY
CALL READY (ICOOR, IERR)
CHECK IF CONTROL VARIABLE IS READY
IND = 0 ?
(NOT READY)
=0(READY)
WARNING CONTROL VARIABLE NOT READY
CALL ONOFF (ICOOR, IERR)
CHECK IF CONTROL VARIABLE IS ON
OFF
ON
SWITCH CONTR. VAR. ON.
ANY CONTROL STRATEGY e.g. OPTIMIZATION
ERRONEOUS DISK OPERATION
STORE NEW OPERATION CONDITION IN DATA BANK FOR REFERENCE
IS OPTIMIZATION CRITERION IMPROVED ?
CALL SRREF (ISOFT, IERR)
CALL SYBAK (ISOFT, IERR)
COMPARE NEW OPERATION CONDITION WITH INITIAL ONES
IS OPTIMIZATION CRITERION BACK TO ITS INITIAL VALUE ?
CALL STBUF (ISOFT , IERR)
RETURN TO OPERATION CONDITION BUF AFTER OPTIMIZATION
CALL FRBUS (N,ISOFT,IERR)
CLEAR BUSY INDICATOR TO FREE CONTROL VARIABLE
EXIT
NOTE:
DATA BANK SYSTEM SUBROUTINES

Figure 2

where

P is the pointer to the first item of the group (first word in the group characteristics),
I is the item number.

DATA BANK HANDLING SOFTWARE SYSTEM

Every process control program has to refer to the data bank. However, one should prevent part or all of the data bank from being destroyed, for example by software bugs. In particular, the disks on which the synchrotron injector data bank is stored have a moving search head and the average access time is around 200 msec; moving the head can almost not be avoided because of the data bank structure (four cylinders or tracks; the first one contains group specifications, the remainder contains individual specifications). Therefore, some retrieving and updating operations need to be optimized to be short in time. As a general rule a programmer should not have direct access to the data bank, but only through an appropriate software system.

The various subroutines are written in machine language with FORTRAN compatible calling sequences. The error indicator is always zero after successful termination of the subroutine.

All those subroutines are modular for simple program structure as is shown in Figure 2.

CONCLUSION

Implementing the data bank has proved very useful, especially during the commissioning of the various process components. For example, the calibration of the components required frequent modification of the definition of the least significant bit, both for control and for acquisition. This could be done in a flexible way through an interactive program displaying the current control variable specification on an alphanumerical display and returning the modification to the data bank.

All control programs have immediately the latest information in hand and need not to be reassembled and recompiled.

Furthermore only a rather small amount of information needs to be loaded in core for controlling part of the process.

However, the two-level structure is a limitation, particularly when vectors using similar elements are given different names for display purposes and also when various process variables only differ because of their name whereas all other group specifications are identical. In this case a three-level structure is more flexible but requires faster disk access.

ACKNOWLEDGMENTS

The author gratefully acknowledges the discussions he has had with H. Van der Beken, L. Burnod, and W. Remmer, which have benefited this work.

REFERENCES

1. Asseo, E., L. Burnod, A. Daneels and E. Sigaud, *Systéme Midi-Console—Solution économique pour l'operation à accès rapide, via ordinateur, d'un processus a paramètres multiples.* CERN/MPS-SI/CO 71-3.
2. Duby, J. J., "Data structures," *Proc. 1970 CERN Computing and Data Processing School*, Villa Monastero, Varenna, Italy, 30 August-12 Septembre 1970, CERN 71-6 (1971), p. 335-404.

Design of a minicomputer network for the automatic determination of amino acid sequences in proteins*

by E. L. BAATZ, B. W. JORDAN, JR., K. J. KING, W. J. LENNON and Z. Z. STROLL

Northwestern University
Evanston, Illinois

BACKGROUND[1–5]

A protein is a simple, nonbranching chain, where any link in the chain is one of approximately twenty different amino acids. The chemical tools available for the determination of the primary structure of a protein (that is, the types of amino acids that are in the protein and the sequence in which they are linked) are relatively few in number and vary greatly in sophistication.

Hydrolysis, boiling a protein sample in a hydrochloric acid solution for roughly a day, completely breaks up the protein by rending every amino acid from its neighbor. By passing the resulting product through an amino acid analyzer, the relative amount of each amino acid in the protein can be determined.

Sequential degradation allows the removal of any amino acid from the amino end of the protein. By repeatedly applying the process and determining which amino acid was removed at every step, it would appear that the primary structure of any protein could be arrived at in a very straightforward manner. However, because the reactions used do not have one hundred percent yield, the sample becomes more contaminated after each amino acid removal. Consequently, the process is limited routinely to sequences of twenty to forty acids. Unfortunately, the typical protein of interest is generally larger, in the range of one to two hundred acids with proteins of up to 2000 acids having been reported.

Proteins can also be fragmented into shorter chains of amino acids by introducing enzymes which break the amino acid chain only between certain pairs of acids. After separating the peptides, each may be treated as a protein in its own right. By fragmenting the peptides, if necessary, chains of sufficiently small numbers of acids can be obtained so that all analytical methods work well.

The primitive techniques above generally allow the unique amino acid sequence of a protein to be determined by collocating the results of two independent experiments with samples of the protein. The general overall procedure follows.

An enzyme is selected to fragment the protein into shorter peptides. The individual peptides are separated using column chromatography techniques. After each sample peptide is judged pure by computing the stoichiometric ratios of its amino acids, the sequence of acids is determined by sequential degradation. Thus the amino acid composition of the protein is known as well as the amino acid sequence of individual component peptides.

The second experiment is a repeat of the first using a different enzyme which fragments the protein into different peptides. A comparison of the amino acid sequences of the peptides obtained from both experiments generally allows the deduction of the complete sequence of the protein.

At present, the above process is done almost entirely by hand, with the aid of a machine for amino acid identification and sequential degradation. It is generally a long process, measured in terms of a few months, consisting of a myriad of exacting steps performed by a skilled technician. There is a growing need to determine protein sequences in research concerning virus related diseases, cell chemistry and evolutionary relationships. For these reasons, the repetitive aspects of these procedures become an obvious candidate for automation.

GENERAL CONSTRAINTS AND PHILOSOPHIES

As Figure 1 shows, the protein analyzer has been designed in six physically separate modules. This was done for three reasons. First, the chemistry and decision making procedures for each module are different and relatively independent. Second, it is desirable to press each complete module into service in order to gain operational experience with it as work continues on the remaining instruments. Finally, modularity seems to lead directly to great flexibility and reliability.

Reliability of the system as a whole must be more than a catchword for two principal reasons. First of all, one means of having a machine procedure be faster than the equivalent human oriented procedure is to have it run around the clock. Such a time schedule suggests that little, if any, operator supervision will be available for long periods of time. Since the system handles a wide range of chemicals, some of them quite toxic, it should not break down in such a manner as to endanger an operator or its surroundings.

* This research is supported by N.I.H. Grant No. 1–R01–GM20048–01–COM

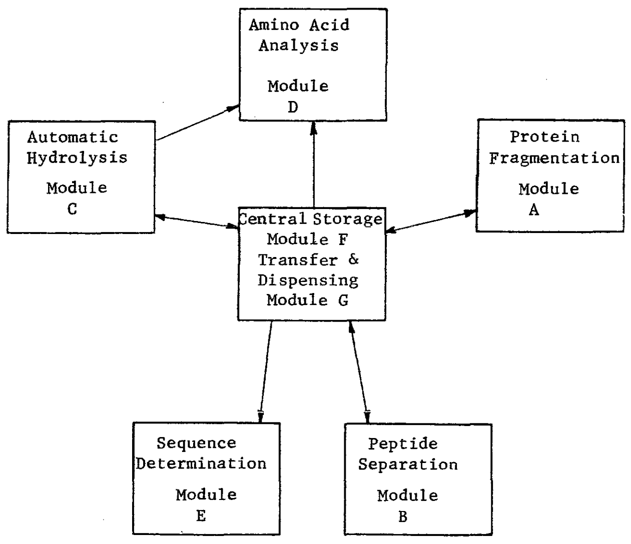

Amino Acid Analysis Module D
Automatic Hydrolysis Module C
Protein Fragmentation Module A
Central Storage Module F Transfer & Dispensing Module G
Sequence Determination Module E
Peptide Separation Module B

Figure 1—Process flow diagram

Second, some of the proteins of interest are rare in every sense of the word. The amount of the sample may be only a score of microliters, and another supply may not be available for years, if ever. Because of these procurement and replacement problems, the system must not accidentally destroy samples.

A system composed of so many complex machines will break down regardless of the amount of care taken in its design and construction. Therefore, it should be fault tolerant in both hardware and software. When something fails, it should fail in a nondisastrous manner. The system should detect the failure, isolate it from the rest of the system, and work around it as much as possible until it is repaired. The modular arrangement of Figure 1 leads to such a system, as shown in the succeeding sections.

All of the decisions in the system are to be made in real time. Where a sample is, what the next step of the analysis is, whether a sample is pure, and many other questions must be answered within various time constraints. The two extremes of the system are represented by the hydrolysis module and the peptide separation module.

Hydrolysis requires activity during half an hour to load the samples into the module and to start the cooking process, followed by simple monitoring for twenty hours, finally followed by cooling and returning the sample to central storage. This is a pattern of moderate activity periods separated by long periods of little or no work.

On the other hand, peptide separation requires continuous observation of the effluent of separation columns. The presence of an amino acid is detected by an increase in the opacity of the effluent at a specific wavelength. Peak sharpening must be done as well as the determination of whether or not the peak is actually two intermixed peaks. If a sample is pure, it must be determined just when to start collecting it, or if it is not pure, it must be diverted for further separation. All those critical decisions must be made against a time scale marked in milliseconds.

The system should be powerful and flexible but not all things to all people. In particular, it is probably impossible to construct a machine that will analyze an arbitrary protein. In addition, it has been observed that even a modest amount of knowledge about the protein under analysis pays dividends in tailoring the general procedures to the particular analysis. The overall intelligence of the system must be high enough that it is capable of receiving instruction from biochemists about custom tailoring the analysis for a particular class of proteins.

The system's initial task will be the determination of the amino acid sequence for varients of the protein cytochrome-c. The system will be able to take advantage of the experience gained by Dr. Emanual Margoliash from the analysis of over twenty samples. When considering the analysis of another class of proteins, it is likely that at most two of the instrument modules will require modification to accommodate new procedures.

A multicomputer architecture provides a superior system over that of a single computer under nearly every criteria mentioned. Primarily, it allows the logical and physical separation of one module from another. This allows simplified hardware and software to be developed, because the needs of a module can be handled in isolation from the rest of the system. Two additional benefits accrue. With the multiplicity of processors, the system is more fault tolerant. One

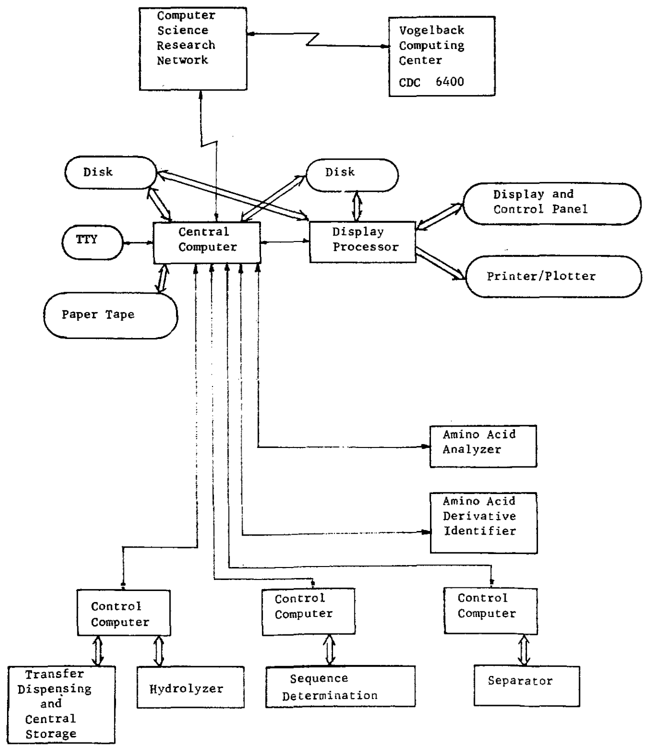


Figure 2—Primary network structure

computer failing still leaves enough intelligence in the system to continue operating in a degraded mode. Since each module has enough computing power to work in isolation from the rest of the system, the benefits of automation are available before the entire system is finished and later can be applied to configurations differing from that of the original system.

HARDWARE

The general organization of Figure 1 is realized by allocating a minicomputer to the control of each module, and having two additional minicomputers for control of devices concerned with the overall information flow of the system. Figure 2 shows this basic structure.

The interprocessor communications are via high speed asynchronous serial lines that run at a rate of between 20K and 153K baud. This means was chosen because of the simplicity and reliability shown by the similar communications paths used by the Northwestern Computer Science Research Laboratory Network.[6,7]

The one process module per computer principle was amended by noting that the hydrolysis module is relatively idle for long periods of time. Therefore, it is felt that its control computer can easily handle activities such as transfer, dispensing, and central storage, which are not time critical.

As Figure 3 shows, reliability considerations greatly complicate the design. All communication paths are duplicated. Every control computer has a watchdog timer and the potential to be restarted from read-only memory. More importantly, there is a control path from two computers to each module. If it is determined that a computer is bad, it can be switched out of the system by a processor at the next higher level and a secondary control computer can take over the affected module. Optimally the effect of the takeover is to have the system run as if nothing has happened. More realistically, the doubly burdened computer will provide degraded service to both modules, but service continues nevertheless. At the worst, one module will merely be maintained in some safe state to avoid damaging the sample or the facilities.

The idea of levels of backup is encountered many places in the analyzer. At the system level, the control computers are at the lowest level, then the central and display processors and finally the operator. Switching out of different parts of the analyzer can take effect only from the next higher level. Therefore, a control computer is removed only by the central computer, and the central computer's tasks are taken over by the display processor only under the direction of the operator.

At the module level, backups also exist. For example, the control of a constant temperature bath is primarily controlled by a closed loop servosystem but is also monitored by the control computer. If the computer decides that the automatic system is misbehaving, it can switch into direct control, effectively closing the system. As a last line of retreat, the loop can be controlled manually.

Experience indicates that hardware fails infrequently inside a central processor. Therefore, the external logic of the system is very primitive, responding to registers set by a control computer. The complicated logic is kept where it is most easily accessible and most easily monitored, in the computer.

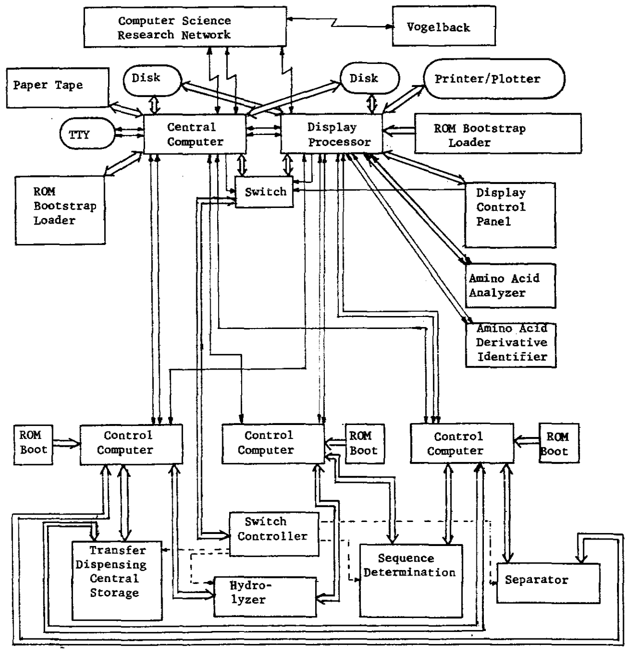


Figure 3—Backup network structure

SOFTWARE

As does the hardware, the software emphasizes that no one part of the system be essential to the operation of the whole. Even though the system is configured as a star shaped network, the central processor does not play a prominent role. Instead it serves mainly as a message switcher, disk controller and system table interlock. The organizing intellect of the system is not localized to one processor, but instead is distributed throughout the system.

A computer's primary responsibility is to completely maintain and manage a particular module. Secondarily it provides support to the system. This is achieved by having the module control software determine when it is idle. The monitoring of the module is left to a much smaller program, and a copy of the system monitor is obtained from the disk. Thus equipped, the computer can make global decisions. Therefore, the coordinating intellect of the system is a potential resident of any machine in it, and the collapse of any particular computer only marginally incapacitates the effectiveness of the entire system.

Whenever a control computer is not immediately concerned with an activity in its module, it first tests the communication lines to the rest of the system. Then its module's queue is checked for any requests from other modules. If there are none, a system queue is checked for nonmodule oriented requests, such as the analysis of some data determined by a module. If no such work exists, diagnostic programs run, and when that is done, decisions are made about which step in the analysis process a sample should progress to.

Since a control computer completely handles the intramodule manipulations, intermodule, time critical actions do not occur as long as queues of samples exist between modules. Therefore, the overall system performance can be task driven on a request basis. That is, if a computer wants something done, it places a request in a disk queue. Some time later, a computer capable of fulfilling that task will pick it up and do it, but possibly only after defining other tasks at a more detailed level. This introduces several levels of tasks in the system, each level being particularly suited to expressing some desired activity in meaningful terms. For example, a task might be a request to Transfer and Dispensing (module G in Figure 1) to pick up a sample from a buffer. Transfer and Dispensing would then break the task into more primitive tasks until it determined how many steps are required to move the transfer head to the sample.

The entire state of the system is kept in great detail on the disk. It keeps not only a past history of each sample and what should be done with it next, but also what is presently being done to it. This means that each control computer frequently sends status information to the disk, which allows all the pieces to be picked up if the control computer of a module goes down.

CONCLUSION

Modern techniques in computer organization promise to produce a reliable, flexible, powerful tool for biochemical research. This system is currently under development by a group at the Technological Institute of Northwestern University under the direction of Dr. Emanuel Margoliash.

REFERENCES

The determination of amino acid sequences:

1. Schroeder, W. A., *The Primary Structure of Proteins*, Harper and Row, New York, 1968.
2. Blackburn, L., *Protein Sequence Determination, Methods and Techniques*, Marcel Dekker, Inc., New York, 1970.
3. Needleman, S. D. (Ed.), *Protein Sequence Determination*, Springer-Verlag, New York, 1970.

Information content of amino acid sequences:

4. Nolan, C., and E. Margoliash, Comparative aspects of primary structures of proteins, *Ann. Rev. Biochem.*, *37*, 727, 1968.
5. Fitch, W. M., and E. Margoliash, The usefulness of amino acid and nucleotide sequences in evolutionary studies. In *Evolutionary Biology* (T. Dobzhansky, M. K. Hecht and W. C. Steere, Eds.) Vol. 4, Appleton-Century-Crofts, New York, 1970, p. 67.

Computer communication:

6. Lennon, W. J., J. T. Spies, and R. C. Barrett, A minicomputer research network. In *Compcon 73 Digest of Papers*, Institute of Electrical and Electronics Engineers, Inc., New York, 1973, p. 191.
7. Barrett, R. C. and W. J. Lennon, A universal serial interface, Spring 1971 DECUS Symposium.

An approach to the optimization of an olefins plant

by S. REITER, D. J. SVETLIK and A. M. FAYON

Mobil Oil Corporation
New York, New York

INTRODUCTION

Mobil Chemical Company has operated an olefins plant in Beaumont, Texas since 1960. This paper discusses the approach taken by Mobil Chemical Company in the optimization of the operation of the olefins plant.

A linear programming (LP) model is used for long and short range planning purposes while open loop on line optimization via an 1800 process control computer is used by plant management for aid in the determination of day to day operating policies.

The heart of the olefins plant is the pyrolysis furnaces. The furnaces yield, from basic feedstocks, the desired olefin products. The processing units downstream of the furnaces separate the furnace effluent stream into the desired high purity olefin product streams.

The olefins plant is capable of processing a large number of feedstocks which range from ethane to naphtha and raffinate. The feeds are obtained primarily from the Mobil Oil refinery in Beaumont. Outside purchased stocks are also available for processing. Each feedstock has associated with it a yield pattern for ethylene and heavier olefins. Depending on market conditions and other factors, at any given time, feedstocks become relatively more or less desirable. Within the yield pattern associated with a given feedstock there is a specific furnace severity, defined by furnace temperature and pressure and hydrocarbon and steam flow, which defines the optimum processing conditions for a specific feedstock.

The two questions raised here, that is;

- what feedstock selection to make? and
- at what conditions selected feedstocks are to be processed?

led to the development and implementation of the systems in use today.

The overall structure of the linear programming model will be reviewed with a discussion of its application to planning decisions.

The hardware and software structure of the 1800 system will be discussed and specific functions associated with it reviewed. The nature of the optimization procedure employed and the structure of the online process model and its adaptive characteristics will also be discussed.

OLEFINS PLANT LINEAR PROGRAMMING MODEL

Structure

The olefins plant LP model is an economic and process model. It consists of linearized representations of the operations of process units with explicit representations of the key constraints associated with these units. Most importantly, it contains representations of the flexibility associated with feedstock purchasing and product sales. A simplified structure of the LP model is shown in Figure 1.

The first operation shown is purchasing. As many as six potential feedstocks may be available for purchase over a given time. Typically the furnace feedstocks available are: ethane, propane, n-butane, isobutane, naphtha, and raffinate. Each of these feedstocks has availabilities associated with it at various prices. In addition to furnace feeds, two additional feedstock availabilities are represented. These are refinery gas, a stream consisting of ethylene and lighter components, and a butadiene concentrate stream, consisting of butene and butadiene. Both these streams are introduced into the process downstream of the furnaces.

The next operation shown is the pyrolysis furnaces. Physically 22 furnaces are available. Due to decoking operations 19 furnaces are usually on-stream at any one time. As many as five yield slates may be associated with a furnace feed stock, as required to linearize the feed stocks yield pattern. Although severity is a multi variable function, a key control parameter is temperature, and in the LP, furnace yields are represented as functions of temperature only. Associated with each feedstock is a furnace stream factor. The stream factor defines the normal mass flow associated with a given feed-stock and is used in defining the maximum furnace availability constraint.

The remaining process operations shown are process gas compression and hot and cold train distillation. Linearized representations of these units are used to develop constraints associated with compressor horsepower requirements, and feed and overhead rate limits on various downstream distillation towers.

The last operation shown is sales. Four olefin product streams, ethylene, propylene, butadiene, and butylene, at

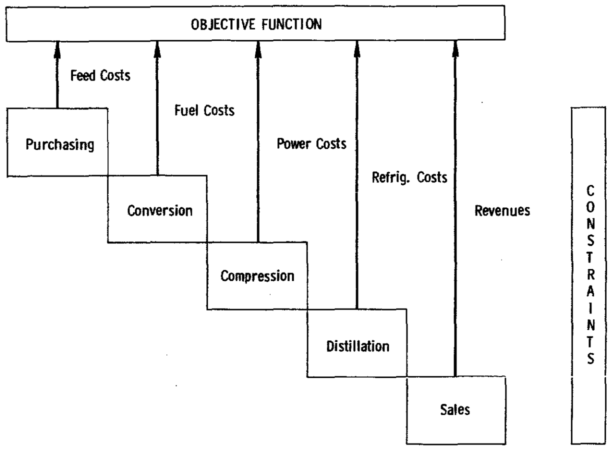


Figure 1—Linear program model structure

one or more pricing levels are represented as well as fuel gas and an aromatics concentrate stream.

Application

The range of applications associated with the LP are:

1. Profit plan and five year plan preparation.
   - Based on anticipated prices and feedstock availabilities a determination of the optimal way to operate the plant for a period of one to five years is developed for budget and profit plan preparation.
   - Based on the above, sales and purchase commitments for periods covering one to five years may be developed.
2. Short Range Planning.
   - Based on predefined feed and product prices optimal feed slates, and product mixes are determined and first pass estimates of furnace severities are established.
3. Equivalent pricing of alternate feedstocks.

   Due to factors, not controllable by Mobil Chemical, planned feedstocks may become unavailable and an alternate feedstock must be selected. The LP model will be used to determine:
   - Which alternate feedstock should be selected;
   - The required availability and price of the alternate feed stock to maintain a constant profit; and
   - The desirability of purchasing outside product streams to meet sales commitments.
4. Effect of debottlenecking on profit.

   The LP is used to assess the overall desirability of implementing recommendations developed from offline simulation studies aimed at debottlenecking.

As indicated above the results of the LP solution provide an indication of the optimal furnace severity to be associated with each feed stock. This severity is a first approximation. It is developed from a linearized economic model which provides adequate results for long and short range planning purposes but which leaves much to be desired in setting actual day-by-day operating conditions. Furnace yields represented in the LP are linearized for typical feedstocks. The effect of feed composition variation is not accounted for. Tower performance is represented by linearized or constant split factors which do not consider tower feed rate or composition variations. Process constraints are expressed as the expected or typical limitation associated with a given unit. Some constraints are implied rather than explicitly represented. As previously indicated furnace severity is equated to operating temperature with hydrocarbon and steam rate and pressure assumed constant.

These approximations are valid for the planning decisions to which LP results are applied. For operating decisions aimed at maximizing plant throughput on a daily or weekly basis a more rigorous representation of the process and its variability or non-linearity is required. It was with the objective of providing such a decision tool that an online control computer system was installed.

OLEFINS PLANT CONTROL COMPUTER SYSTEM

Overview

The justification for the installation of an online control computer was based on improved plant performance achievable through optimization. Since installation of the system additional benefits to operations have been achieved.

The functions performed by the control system can be placed in one of four categories, these are:

- Operations and Management Information Reporting;
- Supervisory Control;
- Operations Support; and
- Optimization.

Before reviewing each of these functions the hardware configuration and system software associated with the control computer will be discussed.

Hardware

Figure 2 is a simple schematic of the computer system hardware. The system consists of an 1800 computer with a 2-mu second core of 32,000-16 bit words. Secondary storage is provided for by a 2-disk storage unit. Operations and systems support input/output facilities are serviced by an IBM model 10 Process Operators Console (POC) and three 1053 typers and a card reader/punch.

Physically, two typers and the POC are in the Olefins plant control room while the main frame and remaining peripherals are in an adjacent computer room.

One typer and the card reader are used primarily by the control group for system maintenance functions. The POC and remaining two typers provide the interface facilities for the Operating Departments' personnel.

The system monitors 362 process input signals and exercises supervisory control over 113 set point stations. With feedbacks, a total of 585 process signals are interfaced with the control system. All input signals are multiplexed and are on a four minute scan cycle. Hardware comparator limit checking is continuously carried out for the detection of out of range signals.

Software

The system operates under TSX. Core is segmented into a fixed and variable core region. Fixed core is approximately 12,000 words and consists of core resident system and control programs and incore tables. Variable core is 20,000 words and is used for applications and control programs with relatively low utilization frequencies.

PROSPRO I is utilized for process I/O interface functions. This includes:

- Process variable scanning;
- Analog to digital conversions;
- Digital to engineering units conversion;
- Control of pulse out signals to set point stations; and,
- Out of limits and signal rate deviation alarm message generation.

POC software operates under TSX. The following functions are provided for:

- Display of selected process variables;
- Change set point and variable min/max limit values;
- Initiation of various log reports; and,
- Initiation of various control system functions, such as the optimization system.

TSX, PROSPRO, and the POC programs are the main system software programs.

Other major software systems, related to applications, include:

- The optimization and process model programs; and
- The furnace start-up/shut-down programs.

These will be discussed under the control system functions they are associated with.

Control System Functions

The four control system functions associated with the olefins plant are: (1) Operations and Management Information Reporting; (2) Supervisory Control; (3) Operations Support; and (4) Optimization.

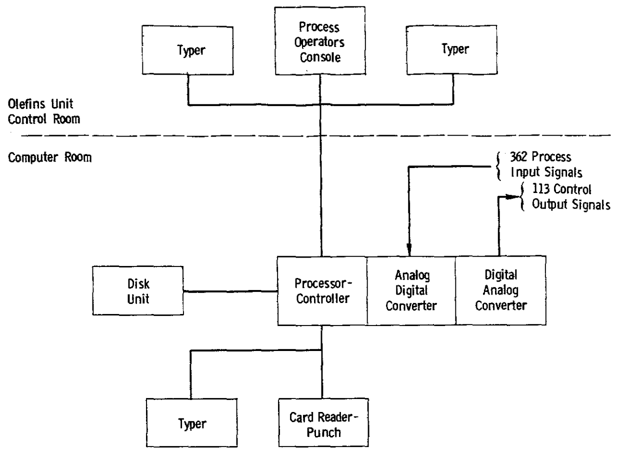


Figure 2—Hardware configuration

1. Operations and Management Information Reporting.

Many control system installations have been justified on the basis of Management Information Reporting. Whatever the justification for a control system, the data collection function performed will always lead to the ability to selectively generate various process reports. Proper transformation of this raw data can provide more useful information to operating personnel.

A number of reports are generated for the operating department and management at the Olefins Plant. These include:

- Furnace log reports;
- Distillation Train reports;
- Compressor logs;
- Plant Material Balance Reports.

The furnace log report will identify for each furnace the feed associated with it, the feed rate, steam to oil ratio, the effluent temperature, the furnace box wall temperature and the quench boiler's outlet temperature. The number of days the furnace has been online and an indication of the degree of coking associated with the furnace are also provide. The material balance report displays the rate and composition of all feed and product streams associated with the plant and an indication of the percent of material balance closure. This report can provide a four or twenty four hour plant material balance.

The other reports contain raw data as well as computed values of key non-measurable parameters associated with the plant sections they are related to. All reports are either generated on a fixed time cycle or on demand via function keys available on the Process Operator's Console.

2. Supervisory Control

Supervisory control functions are currently carried for furnace mass flow compensation and internal reflux control.

Furance flow consists of both hydrocarbon and steam. The steam flow is ratio controlled to the hydrocarbon furnace feed. The flow orifice coefficient on each furnace stream is periodically updated. This assures that measured flows are accurate, and set point mass flows, which are determined via the control system optimization programs, are correctly maintained.

Internal reflux control keeps key towers operating near flood conditions to maximize product recovery.

3. Operations Support

Several programs have been developed to aid the operating department in repetitive tasks. One of these is the furnace start-up/shut-down program.

The Olefins plant consists of 22 cracking furnaces. On the average 19 furnaces will be online at any time while the remaining three are being decoked. The length of online service for any furnace is a function of the feedstock it is associated with and its operating temperature and steam to oil ratio. Typically, within a given week at least one furnace will be taken out of service while another is brought into service as a replacement. The task of smoothly shuting down a coked furnace and simultaneously bringing online a decoked furnace has been automated via a furnace start-up/shut-down program.

Several benefits have been associated with this program, these are:

- Reduced downstream disturbances in process flows during furnace start-up operations;
- Relieving operators from the task of continuously monitoring furnace start-up operations;
- Elimination of rapid coking during start-ups with resulting short furnace runs; and
- Aid in gaining operator acceptance of the control system by demonstrating the system's ability to free them from repetitive time consuming tasks.

4. Optimization

a. The System

The optimization system in use consists of two major programs. A Control Optimization Program, COP, which is an IBM developed general purpose nonlinear optimizer and a Fortran written process model developed by Mobil.

COP is an assembly language program which employs the method of Sectional Linear Programming to solve a non-linear optimization problem. To understand the structure of the overall optimization system employed it is important to briefly review the working of the optimization method employed by COP and its relation to any non-linear system model which is to be operated upon. A more detailed discussion of COP can be found in Reference 1.

Given a set of independent and dependent variables which represent a given state of some system model, the optimization technique employed linearizes the constraints and objective function about this starting point. This is accomplished by developing a differential relation between each dependent or constraint variable and the objective function with each independent or controllable variable. The differentials are developed by making an incremental change in one independent variable and using the model to determine the effect on each dependent variable and the objective function.

These dependent variable and objective function changes are determined for each independent variable in the system.

The differential set of linear relationships that are developed are solved using linear programming techniques to determine the factor by which each independent variable should be incremented or decremented to improve the value of the objective function.

The overall optimization is accomplished in successive steps which move the model from an initial position to an optimal position.

Each step entails the following:

- Linearization—i.e. The development of the differential relationships between independent and dependent variables;
- Formulation and solution of the differential linearized LP problem;
- Calculation of the moves to be associated with each independent variable based on the LP solution; and
- Linearity error correction to determine the actual model position.

In practice, with the olefins plant process model, it has been found that 12 to 15 steps are typically required to move the model from an initial position to a predicted optimum solution. Approximately eight to twelve minutes are required for the completion of each step calculation with 50 percent CPU time available for optimization.

b. The Process Model

The olefins plant process model is a Fortran program consisting of a number of subroutines. The overall structure of the model is shown in Figure 3. There are one or more subroutines provided to stimulate the operation of each processing unit shown in Figure 3.

The first unit shown is the pyrolysis furnaces. The calculation of the yield for any feed stock is based on the conversion of that feed or an indication of conversion. For light feedstocks, Ethane through Butane, volumetric conversion is developed to calculate component yields. For Naphtha and Raffinate, methane yield is used as a measure of conversion to calculate the yield of the remaining furnace effluent components. The feedstock conversion is a correlated function of furnace severity. The severity in turn is a function of hydrocarbon and steam flow, and the furnace exit temperature and pressure. Due to volumetric changes in the pyrolysis furnace the conversion calculation is iterative. The yield curves associated with each feedstock are determined by furnace effluent analysis. Originally, a furnace effluent sampling system was installed, however, due to continuing maintenance difficulties, online yield analysis never proved to be a workable system for updating yield relationships.

In summary the procedure used to compute feedstock yields involves:

First—the calculation of a furnace severity;
Second—the calculation of feedstock conversion;
Third—updating severity with calculated conversion to account for volumetric changes; and,
Finally—the calculation of yields based on conversion for light feedstocks and methane yield for naphtha and raffinate.

In addition to yields, the furnace subroutine calculates fuel requirements and furnace box wall temperature. Box wall temperature is a constraint to the overall optimization.

The next unit shown is the quench boiler. The quench boiler calculation will determine the suction pressure to the first stage process compressor and the steam generation resulting from the furnace effluent. There is a maximum limit on the quench boiler's outlet temperature due to material considerations associated with downstream piping. This outlet temperature limit is another constraint to the optimization program.

The next unit shown is the four stage process gas compressor. Design head capacity curves are utilized to calculate power requirements and discharge pressures for each stage. Second stage suction pressure and overall compressor power requirements are developed as constraints to the optimization.

The fourth stage flash and prefractionator are the next downstream processing units. These units are extremely important to the operation of the hot and cold train distillation towers which follow. The split of C_2 and C_3 Components and the load placed on downstream towers is determined by the separation occurring here. The key unit here is the fourth stage flash. The flash drum was initially simulated with a short cut calculation procedure developed when the system was initially installed. This was a highly tuned calculation which proved to be inadequate in time as the process moved away from operating conditions then in effect. The calculation has been subsequently replaced by a rigorous treatment of the equilibrium and material balance equations that apply. Convergence techniques applied to this calculation have kept the computation time down to a minimum. The prefactionator is simulated by a set of split factors which are periodically updated as process conditions require.

The hot and cold train distillation towers are handled in one of two ways. Critical towers, such as the demethanizer and deethanizer are simulated based on the Fenske, Underwood, Gilliland correlations. This procedure is employed to obtain a better indication of tower performance as effected by feed stream composition variation. Over limited ranges of tower operation, these correlations have given very satisfactory results.

As the plant moves out of the satisfactory range associated

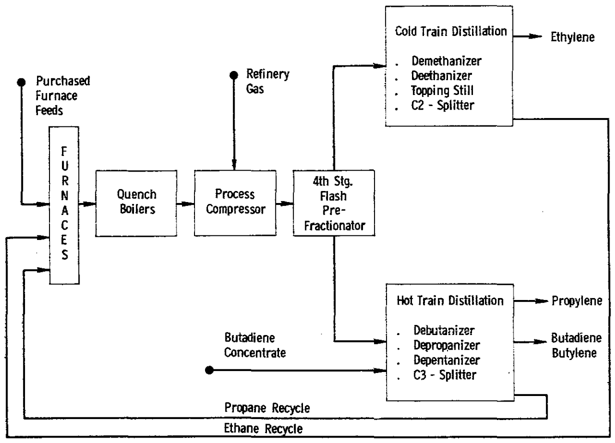


Figure 3—Process model

with these correlations the key component split ratios are adjusted to compensate for this problem. Distillation towers not deemed critical, are simulated by performance factors which are updated as required.

The overall process model calculation is iterative. Initial guesses are made with regard to the ethane and propane recycle streams, in practice these initial estimates are determined from actual process flows.

Three or four model iterations are required before overall convergence is achieved. The time for a model calculation is approximately one minute, with 50 percent CPU time available. The repeated model calculations required by the optimizer accounts for approximately 90 percent of the overall optimization time.

c. Implementation Considerations

An optimization calculation is initiated on an eight hour cycle or on request by operations. When an optimization is initiated, the first step required is the calculation of the plant's present position. This calculation serves two functions:

First—It develops the initial set of independent and dependent variables required by COP; and,
Second—It develops various tuning factors based on measured plant inputs to provide for adaptive tuning in the process model.

The general nature of the tuning carried out is the form:

$$Y = F(X) * \text{Constant}$$

i.e.,

The parameter to be calculated = (a function of known variables) times (a constant)

Since the control system in many cases is monitoring actual plant data associated with both model dependent and independent variables, the actual value of the parameter to be calculated and its associated independent variable

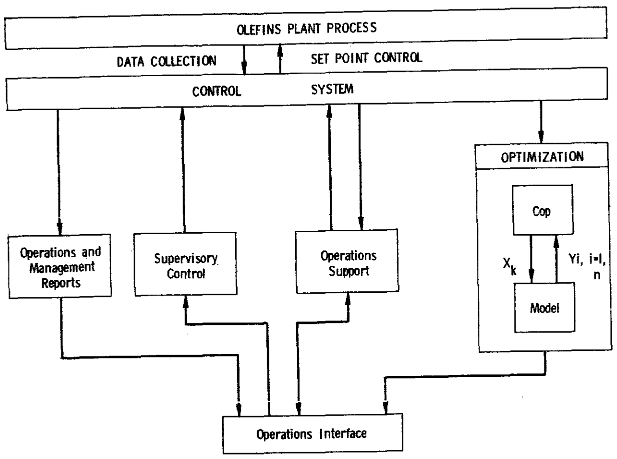


Figure 4—Applications software

values are known. The constant term can then be determined as the ratio of the measured parameter to its calculated value. This constant term is a tuning factor which accounts for process variability with time. This type of tuning is carried out for the following process units:

Furnaces: Used to determine the degree of coking based on heat transfer calculations and is applied to the determination of the furnace box wall temperature.

Quench Boilers: Adjusts clean surface heat transfer coefficients to take into account fouling for steam generation and discharge temperature calculations. In addition, the effect of restricted flow area on the pressure drop calculation is assessed.

Compressor: Develops factors to account for errors in simplified compressor head-flow relations and to take into account degredation in compressor performance due to fouling.

These adaptive coefficients are held constant throughout the optimization. This is an approximation of course, since plant conditions will change as the process is moved from one point of operation to another. In practice, this has not been found to be a difficulty since at any time only small process moves are made and optimizations are carried out on a repetitive cycle. Although analysis of tower product streams are made in the plant, the compositions are not directly available to the computer. This precludes the possibility of carrying out adaptive online tuning of tower performance. Data files containing parameters associated with tower calculations are accessed by operating personnel as required to update these parameters from laboratory analysis of tower product streams.

The optimization system employed provides directional guidance in the setting of controllable variables to improve overall profit. In practice there was found to be considerable hesitancy on the part of the operating department's personnel to except optimizer results when the absolute values of model and optimizer flow predictions differed with measured plant flows. This problem was brought about by material balance errors associated with the plant metering system and unaccounted for material losses.

A real problem presented in the discrepancy between model predicted flows and measured flows was in knowing the correct value to set flow constraints.

The difficulty was rectified by introducing into the model a furnace effluent biasing stream. This stream was calculated by determining the difference in the furnace effluent as determined by the plant material balance metering system and the effluent predicted by the process model. This biasing stream was then held constant throughout the optimization. It achieved the objective of biasing the flow of each component leaving the furnace to the level measured by the plant metering system without affecting the slopes of the non-linear correlations used by the optimization program to develop the required differentials.

As mentioned, some inputs required for model tuning are available via the control system while others must be provided by plant personnel. A large number of parameters are associated with the Olefins plant control system that require frequent updating. Only some of these parameters are related to the model tuning function. In order to place the responsibility of maintaining data integrity with the Operating Department a Control Interface System was developed. This system allows key parameters to be stored in data files and provides for easy access to these files by operating department personnel for either inspection or change functions. These parameters are utilized by various control system programs.

Briefly the information stored relates to:

- Feed compositions for use in T-P-C compensation of furnace feeds;
- Product stream compositions for use in material balance reporting;
- Limits on the ranges of independent and dependent variables used in the optimization;
- Feed, product and utility costs required for the optimization; and
- Furnace feed identifications for use in various plant reports.

Figure 4 presents and overview of all applications software related to the Olefins plant control system. In summary this entails:

- Management and Operations Information Reporting;
- Supervisory Control;
- Operations Support; and
- Optimization.

CONCLUSION

In conclusion, Mobil Chemical Company has developed and implemented a dual approach toward the optimization of its olefins plant.

A linear programming model is used for long and short range planning studies. This is an economic model which represents the flexibility available in feed stock selection and in establishing product mixes. Linearized representations of process units are used.

Benefits related to the use of this planning tool are associated with the quantitative procedures it affords in evaluating alternate choices related to long and short range planning decisions.

An online Control System provides for open loop optimization; the results of the optimization provide operating department personnel with day to day guidance in establishing optimum processing conditions. Additional benefits attributed to the control system are associated with management and operations reporting functions, stabilization of plant flows and various operations support functions.

REFERENCE

1. 1800/1130 Control Optimization Program, User's Manual, IBM Application Manual H20-0351

Computer performance variability

by THOMAS E. BELL

The Rand Corporation
Santa Monica, California

MOTIVATION FOR PERFORMANCE COMPARISONS

During a period when computing on a network is free, users can be very informal about choosing the computer for running their jobs. Certain issues usually dominate this informal evaluation—the convenience of entering jobs, the availability of attractive services, the reliability of the system, and the individual user's familiarity with the system's conventions. These informal evaluations are usually qualitative, but one additional, quantitative characteristic is often included—response time.

Response time, in the context of a computer network, may be defined as the elapsed time for responding to a batch-job run request as well as the more common definition of the elapsed time for responding to interactive requests. On some systems, of course, these two instances are blended together. On other systems the two are distinct, and the values obtained from measuring them are quite different. On two specific computers a user might find that one provides superior batch response time while the other provides superior interactive response time. If everything else is equal and he has no problems in transferring data from one machine to the other, the user would choose the first for batch executions (e.g., running statistical evaluation programs) and the second for doing interactive work (e.g., editing a report).

If installations charge real money for their computing services, another element (money) must be included in an evaluation of alternative computers. Evaluations rapidly lose their informal nature when network users find that their choice of computer determines the amount of money that will remain in budgets for paying their salaries. Personnel time lost due to poor response time, bad conventions, or inadequate services must now be traded off against the costs of avoiding these conditions. Computers that might have been unacceptable prior to charging may become optimal after economics has become a factor in decision-making.

Response time is a single metric, but computer charges are computed from a number of different performance metrics. For example, a bill might be computed from the amount of processor time consumed, the number of cards read, the number of lines printed, the number of tape I/Os and disk I/Os performed, and the amount of core occupied.[1] When each type of resource is separately charged for, one computer may be much cheaper for one type of job but very expensive for another type. If the charge for I/O is relatively low and the charge for CPU time relatively high, the user would be tempted to run I/O-bound jobs on this computer but to submit his "number-crunchers" to another computer. The user without access to a network may be precluded from distributing his work among the available computers in the most economical manner, but the network user has more options to choose from—and more decisions to make. These economic decisions must usually consider each of a number of performance metrics when the rate for each may be different on each of the available computers.

In addition to potentially different rates for each resource, computer centers may employ different functional forms for their billing equations. One of the reasons for such differences is the variety of objectives that they may adopt. Published objectives usually include cost recovery and equitability (or reasonableness) in addition to repeatability.[2,3,4] Other objectives may include limiting load growth to avoid the need for procuring a new machine,[5] biasing users to employ resources available in excess rather than those in short supply,[6] and being able to separate those users with immediate needs (and can afford to satisfy their need) from those who desire cheaper, slower service.[7] The differences in objectives ensure that, as computer centers are increasingly drawn together by networks, users will face more and more different kinds of economic situations to evaluate, and they will find that superficial evaluations will not be adequate for choosing the most appropriate node. Since computational speeds often differ between the machines, rates themselves cannot be compared; the user must compare the total costs of computing through execution of sample jobs.

Either real jobs or synthetic jobs (which use resources in known ways but do no useful computations) can be used in response time and resource charge comparisons. Typically, a user submits a standard job to each of several computers to measure the response time (either interactive or batch) and determine the charges. After finding the values from each candidate node, he picks the one offering him the best ratio of service (response time as well as other services) to cost. This exercise is therefore critical to the user—since it determines the costs he will incur—and to the node—since it determines the amount of load the center will experience.

REPEATABILITY AND VARIABILITY

One of the objectives for charging systems is repeatability, the characteristic of producing the same charge from each of a number of runs of the same job. This same objective of zero variability between runs is often implicitly assumed to have been met by users performing comparisons. They run a single job once on each machine and assume that the resulting performance values accurately represent the machine's performance. This is equivalent to the assumption that there exists zero within-sample variability. Therefore, the variation within samples (e.g., several runs on the same computer) can be disregarded in comparison with the variation between samples (e.g., runs on computer A compared with runs on computer B). If the standard deviation (a measure of variability) of run times on computers A and B were always far smaller than the difference in run times on the two machines, the within-sample variability would clearly not be significant.

Good repeatability would aid users in budgeting their funds for computing as well as help them in making comparisons. Gabrielle and John Wiorkowski[4] suggest that "a variance of no greater than 1 percent is thought to be acceptable." Probably, they mean that the standard deviation of charges should not exceed 1 percent of the mean. This amount of variability is certainly so small that it would interfere very little in realistic comparisons. Perhaps performance variability is unworthy of consideration; some indication of the problem's actual magnitude is necessary to evaluate its importance.

DETERMINING THE MAGNITUDE OF THE PROBLEM

Synthetic benchmarks are being used extensively in performance investigations. For example, Buchholz' synthetic test job[8] has been used by Wood and Forman[9] for comparative performance investigations on batch systems, and Vote[10] has employed his synthetic program in evaluating a time-sharing system. With their increasing use and their documented advantages for certain types of investigations, synthetic jobs are a natural vehicle for determining the magnitude of variability.

Synthetic job

We have used a modification of the Buchholz synthetic test job to determine the magnitude of variability in strictly controlled test situations on IBM, Honeywell, and other manufacturers' equipment. The job (as modified) is written in FORTRAN so that it can be executed on a variety of computers and is structured as follows:

1. Obtain the time that the job was given control and keep the time in memory,
2. Set up for the job's execution,
3. Set up for running a set of identical passes with an I/O-CPU mix as specified on a parameter card.
4. Execute the set of identical passes and record (in memory) the time of each passes' start and finish,
5. Compute some simple statistics from the resultant execution times and print both the times and the values of the statistics.
6. If requested, return to step 3 to repeat the operations for a new I/O-CPU mix,
7. Determine the current time; print out this time and the initiation time.
8. Terminate the job.

As indicated, the job has embedded data collection in the form of interrogations of the system's hardware clock to record the elapsed time between certain major points in the program's execution. When appropriate, the job also determines the accumulated resource usage at each of the major points. The elapsed time within the job can be compared with the time recorded by the accounting system to determine whether initiation/termination time is large enough to require distinguishing between these two measures of elapsed time. In all cases we have observed, the initiation/termination time has been so large that disregarding it would invalidate many conclusions from performance investigations.

The design of the job enables the user to identify the source of certain kinds of variability. Since repeated passes are individually timed, variability that arises from within the period of job execution can be identified. By running the job in a number of different situations other sources of variability can be identified. This job can be used directly to evaluate variability in batch systems, and can be submitted remotely to evaluate variability in remote job environments such as the time-sharing system we investigated.

Interactive responsiveness

While a synthetic job can be used to evaluate the system's response to user-programmed activity, it is not adequate to investigate highly interactive activity like text-editing. The latter type of system is usually investigated by using scripts; the user first makes up a list of commands and then determines the time required for him to complete a series of interactions based on the list of commands. Unfortunately, this approach precludes identifying the source of variability if it arises from only a subset of the commands employed. In addition, the variability of human response is mixed with the variability in computer response. A better approach is to time the response of the computer to each individual command.

The analyst can time these responses with a stop watch as done by Lockett and White,[11] but this technique fails when the computer's response time becomes small. In such situations the human's response time in operating the watch may exceed the computer's response time, and human variability dominates computer variability. In addition, the human often becomes sloppy when large amounts of data are needed because the job becomes tedious. To avoid these problems, we designed and implemented a hardware device to time responses to a resolution of one millisecond.

Analysis approach

Performance can be made to vary by orders of magnitude if a programmer puts his mind to it. By using a computer with a low resolution clock and doing careful programming, a programmer could execute a job that was almost never in control at the time the computer's clock advanced. Thus his job would be charged for using almost no resources. On the other hand, the programmer could leave out the special timing controls and be charged for using hundreds of times as much processing resource. An exercise performed in this way would prove little since any relationship to a normal job stream would be tenuous.

A more useful indication of variability's magnitude would be the lower limit to be expected in more normal situations. Our experiments were directed to this objective and therefore involve simple situations where low variability is to be expected.

Ultimately, all variability could probably be explained. Variations in processing rate could be caused by fluctuations in power line frequency; differences in elapsed time for processing I/O-bound jobs might be explained as variations in the number of I/O-retries; variability in initiation time could sometimes be explained by slight differences in the length of a job control file. However, these circumstances are transitory and usually unknowable to the user. Our tests were designed to represent the best possible situation that a user could realistically expect.

The simplest situation we investigated is one in which no jobs other than the test job are active; the system is initialized at the beginning of the test period to establish that no other jobs will interfere with the experiment. In this simple test the parameterized test job is set to run in either of two modes—as a totally CPU-bound job or as a totally I/O-bound job.

The tests then increased in complexity through controlled multiprogramming situations to uncontrolled, normal operations. Results of the tests were used in simple analyses to indicate the degree of variability that should be expected, but in this initial investigation we made no attempt to employ sophisticated statistical models.

In all cases we used computers that were isolated from any networks, and we usually allowed only controlled on-line activity so that unexpected loading would not occur. In some cases we found that physically disconnecting transmission lines was necessary to achieve an environment that was strictly controlled. Whenever we relaxed our controls, loading became random and variability increased.

TABLE I—Stand Alone Runs—Elapsed Times (All Times in Milliseconds)

Type of Job	Number Passes	Mean Elapsed Time	Standard Deviation
CPU	20	29135	73
CPU	20	29135	73
I/O	20	35165	135
I/O	20	35300	186

EMPIRICAL RESULTS

The most elementary measurement of performance is probably elapsed time, and it is often the one of most interest. Results of elapsed time investigations are therefore presented first, with results involving I/O and CPU metrics following.

Batch elapsed time

The phenomenon of variability in elapsed times under multiprogramming conditions is well-known. When a user's job is run with different mixtures of other jobs, differing amounts of resources are denied it each time it is run; thus the job may execute slowly one time and rapidly the next. One way to decrease this variability (and also decrease the average time) is to ensure that the job of interest is run with the highest priority. The interested network user might therefore occasionally pay to run at a very high priority in order to determine the performance under "best possible" conditions. However, an even better situation is to run stand-alone on the computer. We therefore executed the synthetic job in its simple CPU-bound and I/O-bound versions on an otherwise idle system.

The elapsed time to execute the CPU-bound portion of this job on an IBM 360/65 operating under OS/MVT at The Rand Corporation evidenced no variability within the job that was above the measurement's resolution of 32 milliseconds. (See Table I.) The I/O-bound portion's elapsed time, however, typically varied enough to result in standard deviations (for 20 identical executions) of about .5 percent of the mean (mean of about 35 seconds). This small value indicates that, at least within a stand-alone job, variability is not impressive.

The situation becomes less encouraging when the comparisons are between separately initiated jobs. The statistics of interest now include initiation time, termination time, and average execution time of a pass through the timed loop for each separately-initiated job. The elapsed time for initiating the jobs (the time between initiation as recorded by the accounting system and the time control has passed to the job's code) averaged 8.25 seconds with a

TABLE II—Stand Alone Runs—Channel Times
(All times in Milliseconds)

Run Number	File 1	File 2	File 3	File 4
		Disk		
1	34432	34274	27351	39809
2	34456	34283	27298	39823
3	34429	34257	27334	39816
Mean	34439	34271	27328	39816
		Tape		
4	22356	22533	23328	23087
5	22404	22621	23340	23093
6	22350	22677	23350	23103
Mean	22370	22610	23339	23094

standard deviation of 6.67 percent of the mean. The elapsed time for termination (time user code completes executing until the accounting system records termination) averaged 3.775 seconds with standard deviation of 16.7 percent of the mean.

Although our experiments provided multiple samples of initiation and termination, the sample size of identically run executions was too small for computation of meaningful measures of variability. In two specific instances, however, the average elapsed times to execute the totally CPU-bound portion differed by less than the resolution of measurement. The average execution time for the I/O-bound portion, in two instances of samples of two, changed by 0.4 percent and 1.2 percent. (Allocation of files was done identically in each instance.) Although the within-sample variability appears small for internal portions of a job, the initiation and termination times vary significantly. Therefore, comparisons involving elapsed times should be designed recognizing that the variability of initiation and termination may obscure some results.

On-line elapsed time

An on-line system operating with low priority in a computer would be expected to have variable response, but relatively constant response is usually expected when the on-line system is given a priority only a little below the operating system itself. We ran a series of carefully designed tests to provide an indication of this assumption's validity on our WYLBUR[12] test editor in Rand's normal environment. A heavily I/O-bound activity (listing a file on a video terminal) experienced response time with a standard deviation 23.2 percent of the mean. A heavily CPU-bound activity (automatically changing selected characters) had reduced variability—15.6 percent of its mean. A variety of other editing functions experienced similar variability under a pair of different configurations. The standard deviations as percentages of means ranged from a low of 5.7 percent, to a typical 12 percent, to a high of 30.8 percent. These values were obtained with different levels of user activity on WYLBUR by real users (as opposed to our artificial load for testing), but did not include the variability often introduced by time-sharing systems since the 360/65 was not time-shared.

We executed our standard synthetic job on a Honeywell 615 computer under its time-sharing system to determine variability in this on-line system. The elapsed time to execute each pass through the CPU-bound portion was recorded in order to provide an indication of the variability of response time during normal operation on that system. Each pass could be executed in about 9 seconds in a stand-alone system. However, during normal operation the average was occasionally as large as 597 seconds with the standard deviation exceeding the average. (It was 598 seconds.) Variability of this magnitude can produce highly anti-social behavior by some network users.

I/O activity

Elapsed time is variable in all but the simplest cases, but perhaps the network comparison shopper can depend on charges from a billing system to be within the 1 percent

TABLE III—Multiprogrammed Runs—Channel Times
(All times in Milliseconds)

Run	File 1	Increase	File 2	In-crease	File 3	In-crease	File 4	Increase
		Percent		Per-cent		Percent		Percent
(Disk)								
1	28543	−17.1	37227	8.6	49739	82.0	44329	11.3
1	37678	9.4	48834	42.5	46655	70.7	30638	−23.0
2	27726	−19.5	37366	9.0	47920	75.4	43163	8.4
2	38578	12.0	49977	45.8	45346	65.9	28613	−28.1
3	30896	−10.3	38983	13.7	24687	−9.7	24162	−39.3
4	30200	12.3	47346	38.2	24730	−9.5	24195	−39.2
(Tape)								
3	22698	1.3	22908	1.4	23775	1.9	23570	2.1
4	22873	2.2	23032	1.9	23849	2.2	23675	2.5

Notes:

1. Three jobs were multiprogrammed in each case. The first job was CPU-bound; its results are not represented here. The other two jobs were I/O-bound and were paired as follows:

Run	Second Job Type	Third Job Type
1	Disk	Disk
2	Disk	Disk
3	Disk	Tape
4	Disk	Tape

2. All increases represent changes from the mean times indicated for individual files in Table II.

suggested by the Wiorkowskis. We ran tests on two systems to test this.

Repeated runs of the same stand-alone, I/O-bound job resulted in recorded channel times on a Honeywell Information Systems (HIS) 6050 with ranges considerably less than 1 percent of the means. (See Table II.) When run with one CPU-bound job and only one other I/O-bound job, the channel times changed from the stand-alone values by an average deviation of 2.0 percent for tape files, but 29.2 percent for disk files. (As indicated in Table III, both positive and negative deviations were observed.) In one case, an 82 percent deviation was observed between the charges for the stand-alone run and a three-job multiprogramming case.

All these instances involved simple sequential files. Using this type of file several times with identical jobs usually results in nearly constant performance when considering the metric of I/O requests. I/O counts (Execute Channel Program requests, EXCPs) are reported in IBM's System Management Facilities (SMF) accounting system and seldom vary. However, SMF only reports I/O requests rather than actual I/O accesses. The number of actual I/O accesses normally exceeds the number of requests since channels can execute a number of accesses as a result of a single EXCP. The difference usually becomes dramatic in the Indexed Sequential Access Method (ISAM) where a single request may result in extensive examination of overflow areas.

Processor activity

Processor time appears to be a straightforward metric, but even its definition is open to dispute; major parts of a job's CPU activity may not be logically associated with that job alone. For instance, the rate of executing instructions may be reduced as a result of activity on a channel or another processor in the system. In addition to the definitional problem, accounting systems are often implemented in ways that users feel are illogical. These problems are seldom of importance when a job runs alone in the system, but may be critical during multiprogramming or multiprocessing. Repeated runs of our test job on an IBM 360/65 did not indicate any meaningful variability when the CPU-bound job was run stand-alone, but the I/O bound job, in two cases, provided CPU times of 29.4 seconds and 28.7 seconds (a difference of 2.4 percent of the mean). The I/O variability, however, does not appear critical because this job ran, stand-alone, for an elapsed time of approximately 780 seconds; the difference of 0.7 seconds is therefore less than 0.1 percent of the elapsed time. Under multiprogramming results vary more.

Although the CPU charges should not vary when the job is run multiprogrammed rather than stand-alone, our results indicate that the reported charges contained both a biasing element and a random element. The charges for the CPU-bound job went up from 583 seconds (stand-alone) to 612 seconds (when run with the I/O-bound job) to 637 seconds (when run with a job causing timer interrupts every 16.7 milliseconds) to 673 seconds (when multiprogrammed with both the other jobs). The changes in CPU charges are clearly dependent on the number of interrupts the system handles for other jobs on the system. The largest CPU charge observed in this series of tests with the CPU-bound job was 16 percent over the stand-alone charge, but larger biases can be obtained by running more interrupt-causing jobs simultaneously. In one particularly annoying case, the author observed a production job (as opposed to a synthetic one) whose CPU charges differed between two runs by an amount equal to the smaller of the two charges—a 100 percent variation!

I/O-bound jobs often experience CPU charge variability of equal relative magnitude, and users with I/O-bound jobs have come to expect 30 percent variations in their charges. These problems are not unique to IBM equipment. We found the same sorts of variability when running on a Honeywell 6050 processor. With only a single processor active, we observed processor times that (with two other jobs active) increased up to 7.2 percent over the stand-alone average charge. More jobs and more processors increase the variability.

INTERPRETATION

The results of performance tests indicate conclusively that something is varying. Some of the reported variability is undoubtedly due to the reporting mechanisms themselves. For example, the CPU time reported by IBM's SMF in an MVT system attributes the processing time for I/O interrupts to the job in control at the time of the interrupt. Since that job may be any one running on the system (the job that caused the I/O to be executed or an unrelated one), CPU charges are dependent on which jobs are concurrently executing as well as the micro-level scheduling of these jobs. This scheme had the advantage of accounting for most of the CPU activity at reasonably low cost, but it often reported a number whose meaning was obscure. (IBM's new virtual system reduces the reported variability by not reporting the I/O interrupt-handling time.)

The entire amount of variability cannot be attributed to reporting mechanisms; the basic processes are clearly subject to variability-causing phenomena. Some of these can be identified subsequent to a test and employed to design better-controlled tests in the future. For example, in some of our tests we could have reduced variability by using only old files. New files are allocated by the operating system in a manner that depends on previous file allocations; using only old files would ensure that physical file positions would be identical for each test run and result in reduced variability. However, employing this strategy would preclude determining how well the operating system performed file allocation. In general, the more realistic the desired test, the more variability the analyst must accept.

Even if a test is run with good performance reporting in a very tightly controlled environment, performance must be considered a random variable. If files are allocated prior to testing, random I/O errors still may occur. Rotating devices do not possess ideally constant, known rotational velocities. In addition, micro-level sequencing in a processor is often dependent on the precise start time of user jobs and the entry time of system support modules. The ability to explain these effects after the test does not imply that they are knowable before the test.

Comparisons of elapsed times or charges between computers on a network cannot depend on variability being within the desired few percent. Even in the simplified situations reported in this paper the variability was often large enough to preclude single-sample evaluations from being dependable. If a user intends to do significant computing after choosing a node, he should ensure that his evaluation reflects this reality. Further, he should occasionally check the environment on the network to see whether charges or response time have changed enough to justify a change in his workload allocation scheme.

REFERENCES

1. Hall, Gayle, "Development of an Adequate Accounting System," New York, Share, Inc., Computer Measurement and Evaluation—Selected Papers from the SHARE Project, Vol. 1, 1973, pp. 301-305.
2. Kreitzberg, C. B. and J. H. Webb, "An Approach to Job Pricing in a Multi-programming Environment," *Proceedings Fall Joint Computer Conference*, 1972, pp. 115-122.
3. Young, J. W., "Work Using Simulation and Benchmarks," New York, Share, Inc., Computer Measurement and Evaluation—Selected Papers from the SHARE Project, Vol. 1, 1973, pp. 286-292.
4. Wiorkowski, G. K. and J. J. Wiorkowski, "A Cost Allocation Model," *Datamation*, Vol. 19, No. 8, August 1973, pp. 60-65.
5. Bell, T. E., B. W. Boehm, and R. A. Watson, "Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort," The Rand Corporation, R-549-PR, November 1972. (Also *Fall Joint Computer Conference 1972*, pp. 1141-1154.)
6. Watson, R. A., *Computer Performance Analysis: Applications of Accounting Data*, The Rand Corporation, R-573-PR, May 1971.
7. Nielsen, N. R., "Flexible Pricing: An Approach to the Allocation of Computer Resources," *Proceedings Fall Joint Computer Conference*, 1968, pp. 521-531.
8. Buchholz, W., "A Synthetic Job for Measuring System Performance," *IBM Systems Journal*, Vol. 8, No. 4, 1969, pp. 309-318.
9. Wood, D. C., and E. H. Forman, "Throughput Measurement Using a Synthetic Job Stream," *Proceedings Fall Joint Computer Conference*, 1971, pp. 51-55.
10. Vote, F. W., *Multiprogramming Systems Evaluated Through Synthetic Programs*, Lincoln Laboratories (MIT), ESD-TR-73-338, December 1973.
11. Lockett, J. A. and A. R. White, *Controlled Tests for Performance Evaluation*, The Rand Corporation, P-5028, June 1973.
12. Fajman, R., and J. Borgelt, "WYLBUR: An Interactive Text Editing and Remote Job Entry System," *Communications of the ACM*, Vol. 16, No. 5, May 1973, pp. 314-322.

On measured behavior of the ARPA network*

by LEONARD KLEINROCK and WILLIAM E. NAYLOR

University of California
Los Angeles, California

INTRODUCTION

The purpose of this paper is to present and evaluate the results of recent measurements of the ARPA network. We first discuss the tools available for performing these measurements. We then describe the results of a particular experiment, which consisted of data collection over a continuous seven day period. The measured quantities included input traffic, line traffic, and message delays. This data is discussed in terms of network behavior and compared to analytic models. Lastly, we consider some implications and tradeoffs derived from these measurements which provide insight regarding the performance of computer networks.

The ARPANET is now more than four years old.[1-8] However, the network did not become generally useable until the middle of 1971 when the HOST-to-HOST protocol[5] was finally implemented at most of the sites connected to the network at that time. Currently, the network consists of approximately 40 switching computers (the IMPs and TIPs) and approximately 50 HOST machines attached to these switching computers as shown in Figure 1 (this map corresponds to the network configuration as of 1 August 1973; we use this particular map since it gives the network topology which existed at the initiation of our experiment; a 39th site had just been installed in the network by BBN for test purposes and thus does not appear in Figure 1). We notice that the ARPANET spans the United States, crossing over to Hawaii by means of a 50 KBPS (kilobit per second) satellite channel and extends to Europe by means of a trans-Atlantic 7.2 KBPS satellite channel. From October of 1971, the traffic and use of the network has been growing exponentially at a phenomenal rate, slowing down a bit toward the end of 1973; this traffic growth is shown in Figure 2 on a log-linear scale.[14] In this paper we examine the details of that traffic flow.

The ARPANET began as an *experimental* network and has since grown into a powerful tool for resource sharing. The essence of an experiment is measurement, and it is this aspect of the ARPANET which we wish to discuss herein. Can we, in fact, determine what is going on within the network? The answer is an emphatic yes, if we restrict ourselves to the behavior of the communication subnetwork which provides the message service for the user-HOST systems. Early on, during the days when the ARPANET was still a concept rather than a reality, we were careful to include in every specification of the network design the ability to monitor network behavior with the use of specific measurement tools. This paper deals with a description of those tools and how they have been used in a particular experiment designed to elucidate the behavior of traffic in the ARPANET.

Among the various centers in the network are two which are deeply concerned with measurements; the Network Control Center (NCC), at Bolt, Beranek and Newman, Inc. (BBN), and the Network Measurement Center (NMC) at the University of California, Los Angeles (UCLA). The experiment we describe below was designed, conducted and interpreted by the UCLA-NMC research staff.

At this point, it is perhaps helpful to review a few of the network parameters which affect traffic flow in the ARPANET.[9] All traffic entering the network is segmented into messages whose maximum length is 8063 bits. These, in turn, are partitioned into smaller pieces called packets which are at most 1008 bits long (a maximum length message, therefore, will be partitioned into eight packets, the last of which has a maximum length of 1007 bits). As messages enter the network from the HOSTs they carry with them a 32 bit "leader" which contains the addressing information necessary for delivery to the destination. Incoming messages also carry a small number of "padding" bits for word boundary adjustment between the IMP word size of 16 bits and various HOST word sizes. Packets are transmitted through the network with some addressing and control information which adds 168 bits to their transmitted length, while the packet overhead for storage within an IMP is 176 bits. The packets make their way through the network individually and are passed from IMP to IMP according to an adaptive routing procedure; in each IMP-to-IMP transmission an acknowledgment is returned if the packet was accepted; when possible, these acknowledgments are piggybacked on return traffic. The packets of a multipacket message are reassembled at the

* This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAHC-15-73-C-0368.

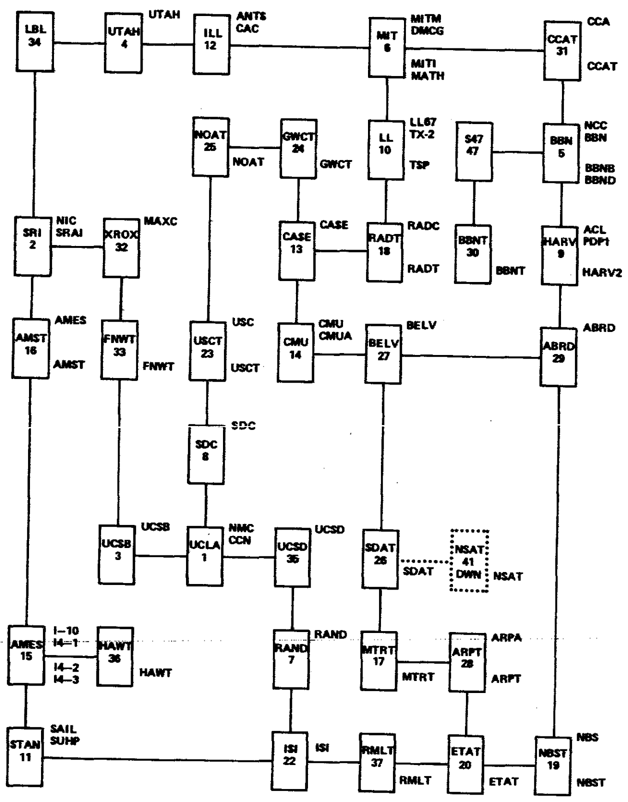


Figure 1—Logical map of the ARPANET (August 1, 1973 0834 PDT)

destination IMP before they are delivered to the destination HOST. When a message proceeds in its transmission to the destination HOST, a special control message (known as a Request For Next Message—RFNM) which acts as an end-to-end acknowledgment is returned from the destination IMP to the source HOST. The IMP itself buffers packets as they pass through the network and has the ability to store approximately 77 packets at most. Except for the channel connecting AMES to AMST (which is 230.4 KBPS) and the Atlantic satellite link (which is 7.2 KBPS) all lines in the network are 50 KBPS, full-duplex channels (as of August 1973).

In the following section, we describe the network measurement tools. Following that, we give details of a recently performed experiment and present its results in graphic form. We also include a section in which a mathematical model for delay is developed and the results of that model prediction are compared with measured network delays.

MEASUREMENT TOOLS

In this section, we describe the means by which this and other measurements are performed. In order to evaluate the performance of the network, several measurement tools (as originally specified by the UCLA-NMC) were implemented as part of the first IMP program (and have been slightly modified throughout the developmental stages of the ARPANET). These tools, which execute in each IMP's "background" mode, may be used selectively at the various network nodes under program control. Upon request, they collect data regarding their node, summarize these data in special measurement messages, and then send these messages to a collection HOST (normally UCLA-NMC). We have, therefore, developed at UCLA-NMC the capability for control, collection, and analysis of the data messages. Below, we describe these network measurement tools.

Trace

Trace is a mechanism whereby messages may be "traced" as they pass through a sequence of IMPS. Those IMPs whose trace parameter has been set will generate one trace block for each marked packet (i.e., a packet with its trace bit set) which passes through that particular IMP. (An "auto-trace" facility exists by which every nth message entering the network at any node may be marked for tracing.) A trace block contains four time stamps which occur when: (1) the last bit of the packet arrives; (2) the packet is put on a queue; (3) the packet starts transmission; and (4) the acknowledgment is received (for store and forward packets sent to a neighboring IMP), or transmission is completed (for reassembly packets sent to a HOST). (Time (1) corresponds to the time at which storage is actually allocated to the packet

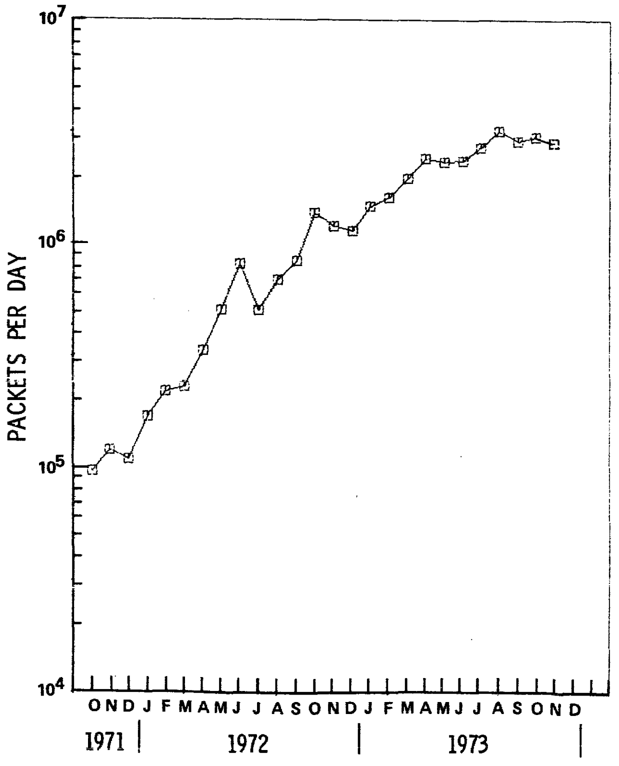


Figure 2—Long term traffic growth

rather than to the input source. Time (4) corresponds to the time at which the storage for the packet is returned to the free pool after successful transmission.) Also contained in the trace block are the length of the packet, an address indicating where the packet was sent, and the IMP header (which consists of the source and destination addresses and several other pieces of control information).

Accumulated statistics

The accumulated statistics message consists of several tables of data summarizing activity at a network node over an interval of time (ranging from 25.6 msec to some 14 minutes) which is under program control. Included in the accumulated statistics is a summary of the sizes of messages entering and exiting the network at the set of real (as opposed to fake, i.e. IMP-simulated) HOSTs connected to that IMP. The message size statistics include a histogram of message lengths (in packets) for multipacket messages and a log (base 2) histogram of packet lengths (in words) for all last packets (i.e., a count is recorded of those packets whose length, in data bits, is from 0 to 1, 2 to 3, 4 to 7, 8 to 15, 16 to 31, or 32 to 63 IMP words in length). Also included is the total number of IMP words in all the last packets, and the total number of messages from each HOST (real and fake), and the total number of control messages (RFNM, etc.) to each HOST.

A row of the global traffic matrix is contained in each IMP's round-trip statistics. These contain the number of round-trips (message sent and RFNM returned) sent from the probed site to each site, and the total time recorded for those round-trips. These statistics are listed for each possible destination from the probed site.

For those channels connected to the probed site, we have the channel statistics. These consist of: (1) the number of hellos sent per channel (channel test signals); (2) the number of data words sent per channel; (3) the number of inputs received per channel (all inputs: data packets, control packets, acknowledgments, etc.); (4) the number of errors detected per channel; (5) the number of "I-heard-you" packets received per channel (response to hello); (6) the number of times the free buffer list was empty per channel; and (7) log (2) histograms of packet length, in data words (one histogram per channel).

Snapshots

Snapshots give an instantaneous peek at an IMP. The snapshot records several queue lengths as well as the IMP's routing table. The HOST (real or fake) queue (normal and priority) lengths appear in each snapshot message. Also included is information about storage allocation: the length of the free storage list, the number of buffers in use for reassembly of messages, and the number of buffers allocated to reassembly (but not yet in use). Snapshots also include the IMP routing table and delay table. Entry *i* in the routing table contains the channel address indicating where to send a packet destined for site *i*. A delay table entry consists of the minimum number of hops to a site, and the delay estimate to reach a site.

Artificial message generation

In addition to the above instrumentation package built into each IMP, we have the capability to generate artificial messages. This message generator in any IMP can send fixed length messages to one destination at a fixed or RFNM driven interdeparture time. Together with the generation facility there exists a discard capability in each IMP. Several message generator/acceptor pairs have been implemented for a subset of the HOSTs on the network as well. These are extremely useful for experimentation, but we will not attempt to discuss them in this paper.

Control, collection, and analysis

The above-mentioned measurement and message generation facilities are controlled by sending messages to the "parameter change" background program in the IMPs. We have constructed a set of programs which, after an experiment is specified, automatically format and send the correct parameter change messages to initiate that experiment. In order to be able to send these messages, it was necessary to modify the system code in the NCP to bypass the normal HOST-to-HOST protocol.[5] The bypass was then used as the means of collecting the measurement messages as well, since these too do not adhere to HOST-to-HOST protocol. After a message is received over this mechanism, it is stored in the file system at UCLA-NMC. Reduction and analysis of the data is accomplished by supplying specific subroutines for a general driver program; the data analysis is currently done on the UCLA 360/91.

Status reports

In addition to the above tools, which are mainly for experimental use, the NCC has built into the IMPs a monitoring function called "status reports."[10] Each IMP sends a status report to the NCC HOST once a minute. Contained in the status report are the following: (1) The up/down status of the real HOSTs and channels; (2) for each channel, a count of the number of hello messages which failed to arrive (during the last minute); (3) for each channel, a count of the number of packets (transmitted in the last minute) for which acknowledgments were received; and (4) a count of the number of packets entering the IMP from each real HOST. These status reports are continually received at the NCC and are processed by a minicomputer which advises the operator of failures in the network and creates summary statistics.

Let us now address ourselves to the experiment itself.

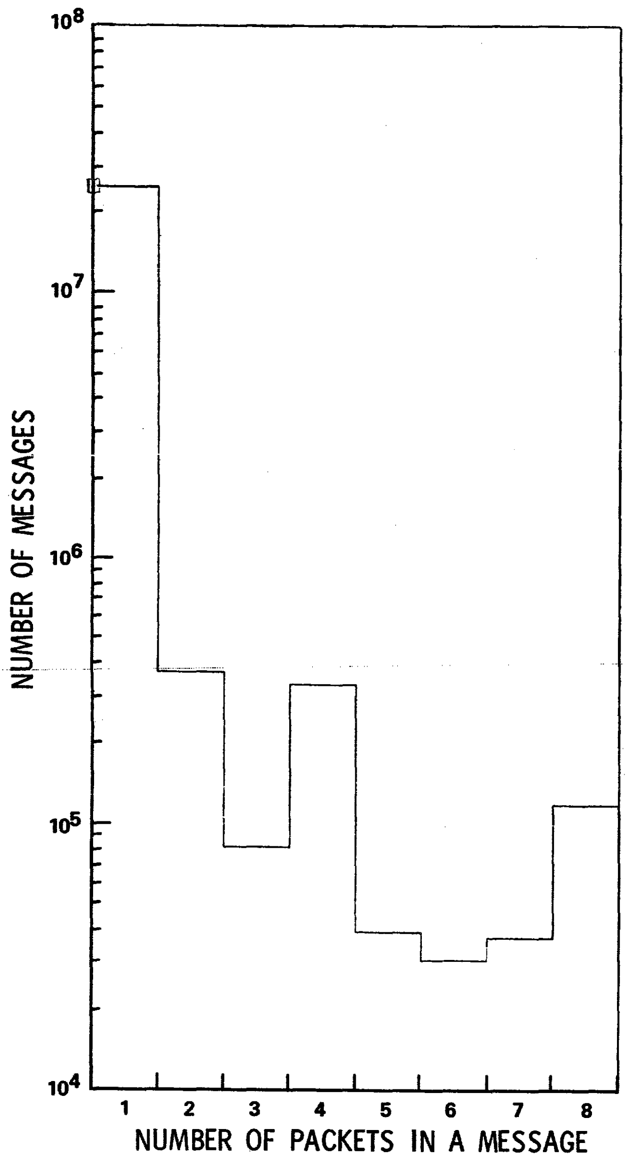


Figure 3—Histogram of HOST message length in packets

THE EXPERIMENT DESCRIPTION AND RESULTS

Experiment description

The purpose of this experiment was to observe the traffic characteristics of the operating network. These characteristics include: (1) message and packet size distributions; (2) mean round-trip delay; (3) mean traffic-weighted path length; (4) incest (the flow of traffic to and from HOSTs at the same local site); (5) most popular sites and channels; (6) favoritism (that property which a site demonstrates by sending many of its messages to one or a small number of sites); and (7) channel utilization. We consider this data to have more than just historical significance. In particular, there are several network parameters whose values were chosen prior to the actual network implementation and which deserve to be reevaluated as a result of the measurements reported here. Among these parameters are: packet (and therefore buffer) size, number of buffers, channel capacity, single/multiple packet message philosophy, etc.

To observe the traffic characteristics, we gathered data over a continuous seven-day period from 8:36 on 1 August 1973 through 17:06 on 7 August 1973. The network configuration during this period is shown in Figure 1. (A teletype-compatible network map containing similar information may be generated from an updatable NMC survey of the network.) The experiment consisted of sending accumulated statistics messages to UCLA-NMC from each site in the network at intervals of approximately seven minutes. The

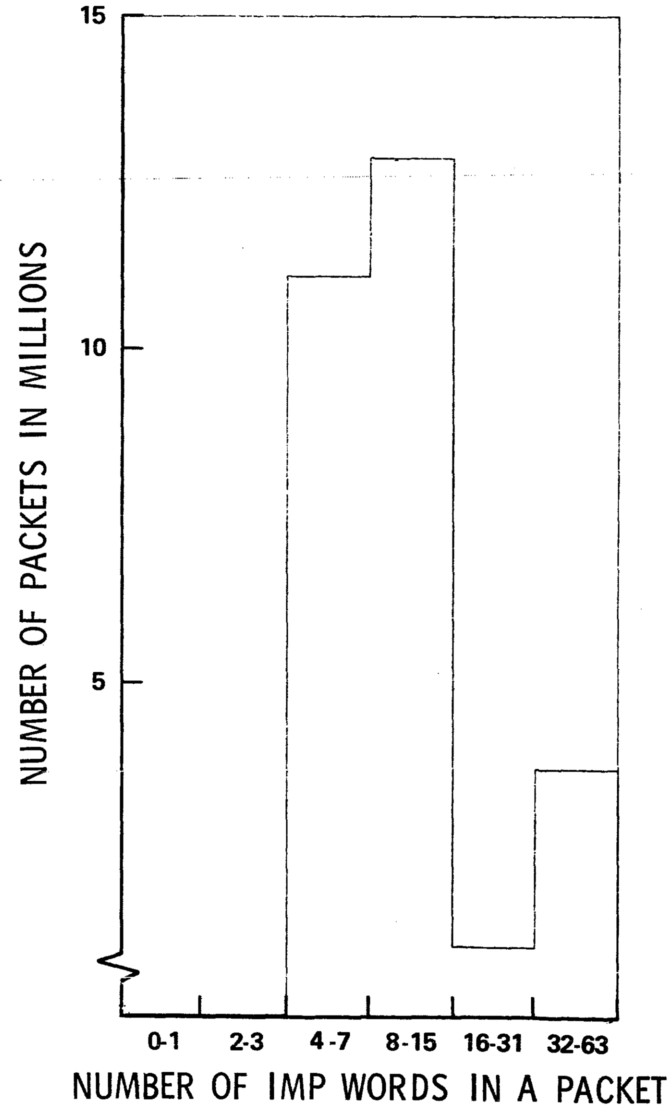


Figure 4—Histogram of packet length in words

data were subsequently processed, and the general results appear below.

Measured results

During the seven days a total of some 6.3 billion bits were carried through the network by some 26 million messages. This means that on the average the entire network was accepting some 47 messages per second and carrying roughly 11500 bits per second among HOST computers. The HOST messages were distributed in length as shown in Figure 3, and from these data, we observe a mean of 1.12 packets per message! Moreover, the mean length of a message is 243 bits of data! These facts indicate not only are there very few multipacket messages, but also that most single packet messages are quite short. This latter fact is borne out in the log (2) histogram of packet length for packets entering the network from the HOSTs as shown in Figure 4; the mean packet length is 218 bits of data.

The small message size has an impact on the efficiency of storage utilization. This may be seen by defining the buffer utilization efficiency as follows:

$$\eta = \frac{\bar{l}_p}{L+H}$$

where

$\bar{l}_p$ = the mean packet length,
L = the maximum length of data in a packet, and
H = the length of the packet storage overhead.

Using the measured value of $\bar{l}_p = 218$ bits, and the constants $L = 1008$ bits and $H = 176$ bits, we have a measured buffer utilization efficiency of .184!

There exists a buffer length which yields an optimal buffer efficiency for a given message length distribution, as shown by Cole;[11] this calculation assumes an exponential message length distribution (which we shall adopt). In the packeting of messages into L bit pieces we have truncated the exponential message length distribution at the point L, thus giving a mean packet size of

$$\bar{l}_p = \bar{l}[1 - e^{-L/\bar{l}}] \qquad (1)$$

where $\bar{l}$ = the mean message size (exponential). This gives $\bar{l}_p = 239$ bits when the value of $\bar{l} = 243$* is used in Eq. (1) and which in turn yields an efficiency of .202. (The fact that $\bar{l}_p = 239$ is greater than the measured $\bar{l}_p$ means that the actual distribution weights shorter messages more heavily than the exponential distribution.) Since $\bar{l}_p$ is significantly less than L, the truncation at L does not cause a large accumulation of packets whose length is L bits; we see this from the moderate number (12.9 percent) of maximum length packets in Figure 4.

The optimal value for buffer size L_0 is obtained by solving the following equation:

$$e^{-L_0/\bar{l}}[L_0 + H] - \bar{l}[1 - e^{-L_0/\bar{l}}] = 0$$

Using $\bar{l} = 243$, and $H = 176$ we obtain the optimal buffer size of $L_0 = 244$ bits which yields a maximum efficiency of .366 for this overhead. Thus, based upon this particular week's measured data, (which is supported by previous and later measurements), we find that the maximum efficiency can be increased significantly by reducing the packet buffer size to roughly one-quarter of its current size.

The measured mean round-trip* message delay for the seven-day period was approximately 93 milliseconds. Indeed, the network is meeting its design goal of less than 200 milliseconds for single packet messages. Thus, as desired, the communication subnet is essentially transparent to the user, so far as delay is concerned. The principal source of delay seen during a user interaction comes both from his local HOST and from the destination HOST on which he is being served. Major contributors to the small network message delay are the small message size and the fact that a significant number of messages traverse very short paths in network.

We shall return to a discussion of delay in the next section. For now, let us study the traffic distribution and the source of short paths, incest, favoritism, etc. From Reference 12 we know that the mean path length (in hops—i.e., number of channels traversed) may be calculated by forming the ratio of the total channel traffic to the externally applied traffic. This gives a value of 3.31 hops. Moreover, we may form a lower bound on the average path length by assuming all traffic flows along shortest paths; this gives a value of 3.24 hops, showing that indeed most of the traffic follows shortest paths. The (uniformly weighted) path length (average distance) between node-pairs is 5.32 as can be calculated directly from the topology shown in Figure 1. The difference between these measures of path length suggests that network users tend to communicate with sites which are nearby. This is surprising since distance in the network should be invisible to the users! This phenomenon may be explained by examining how much traffic travels over paths of a given length (in hops) as shown in Figure 5. Observe that a surprisingly large fraction (22 percent) of the traffic travels a distance of zero hops and is due to (incestuous) traffic between two HOSTs connected to the same IMP; after all, the IMP is a very convenient interface between local machines as well. Also note that 16 percent of the network traffic travels a hop distance of one; the major portion of this (13 percent of the total) is due to communication between AMST and AMES (this too is incestuous in spirit). This curve fails to account for the number of site-pairs at a given distance. For the topology

* A truncation effect occurs before messages enter the ARPA network as well. Hence the measured mean message length is actually the mean taken from the actual distribution truncated at 8063 bits (8 packets). Assuming that messages are exponentially distributed we may solve an equation similar to Eq. (1) to obtain the untruncated mean message length; this computation yields 243 bits, the same as the truncated mean message length.

* Round-trip delay is measured by the IMPS and is the time from when a message enters the network until the network's end-to-end acknowledgment in the form of a RFNM is returned.

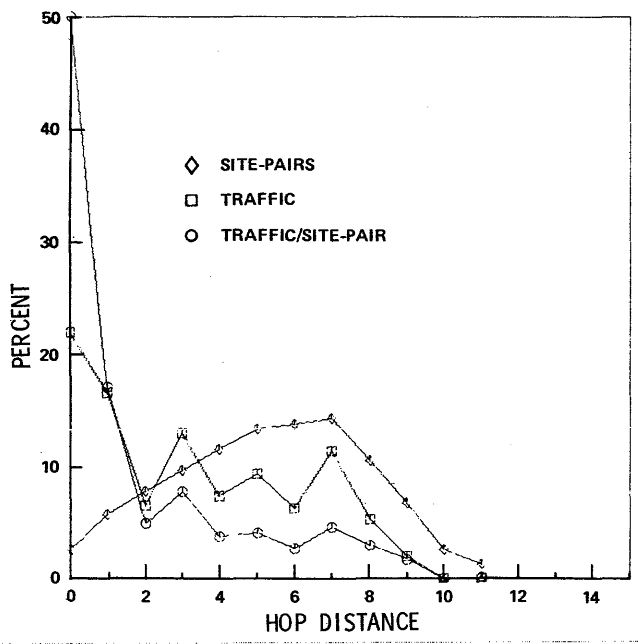


Figure 5—Distance dependence of traffic

existing during this experiment, it can be seen that the following list of ordered pairs (x, y) provides the distribution of site-pair minimum distances (where x=hop distance and y=number* of site-pairs at this distance): (0,39), (1,86), (2,118), (3,148), (4,176), (5,204), (6,210), (7,218), (8,160), (9,102), (10,40), and (11,20). No sites are more than 11 hops apart. This data is also plotted in Figure 5. Note that more sites are at a distance of 7 than any other distance (with the average distance equal to 5.32 as mentioned above). (In a network with N nodes and M full-duplex channels, the first two entries on the list must always be $(0, N)$, $(1, 2M)$.) With this information, we may "correct" our curve by plotting the ratio of the number of messages sent between site-pairs at a given distance to the number of site-pairs at that given distance; see Figure 5 again. The ratios are normalized to sum to one. If the traffic were uniformly distributed in the network, then the resulting curve would be a horizontal line at the value 8.3 percent. We note that an even larger fraction

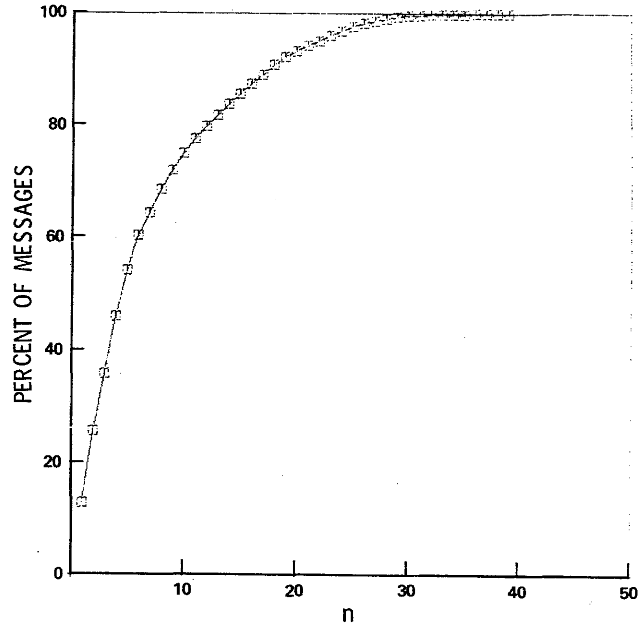


Figure 7—Busy source distribution

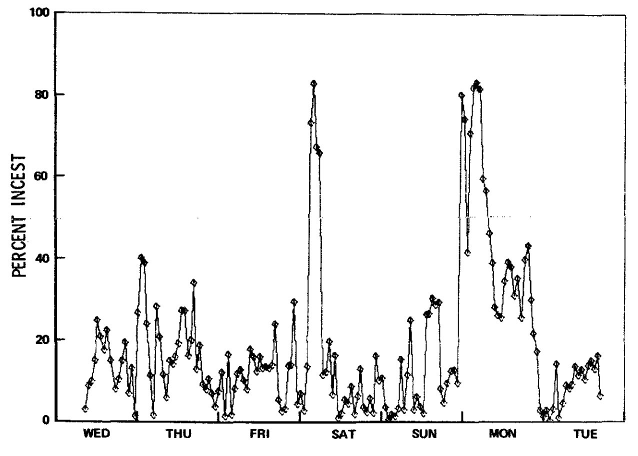


Figure 6—Incest

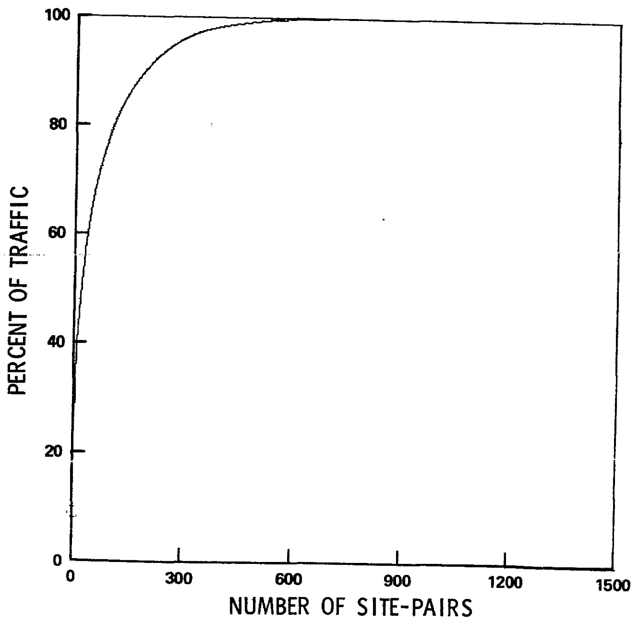


Figure 8—Busy site-pair distribution

* We consider site pairs as ordered pairs; thus, the pair (MIT, UCLA) is distinct from (UCLA, MIT). This is natural since the traffic flow is not necessarily symmetrical. The (important) special case of (SITE i, SITE i) counts as one "pair".

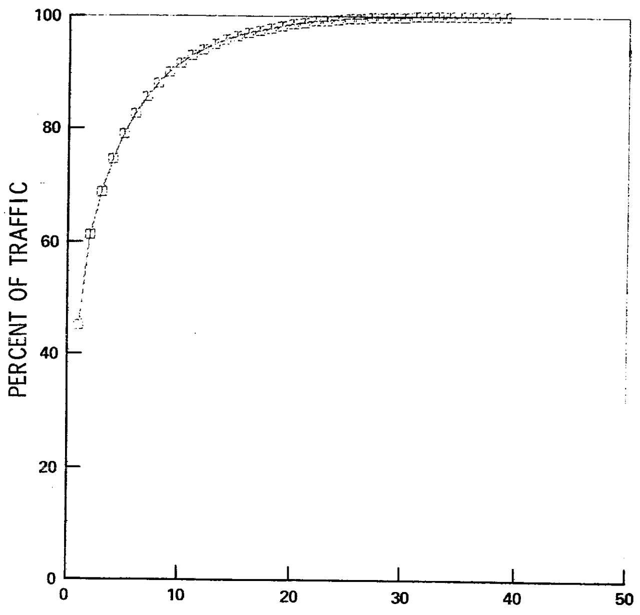


Figure 9—Distribution of traffic to favorite destinations

of the traffic is now identified with distance zero. At distances 2, 3, . . ., 9, we now see a better uniformity than earlier. The last effect which contributes to the remaining non-uniformity is the location of the large traffic users (e.g., ILL) and large servers (e.g., ISI). In Figure 6, we display the percent of incest in the network during each hour* of the experiment. Note that incest accounts for over 80 percent of the traffic during certain hours (the weekly average is 22 percent), peaking in the wee hours of the morning.

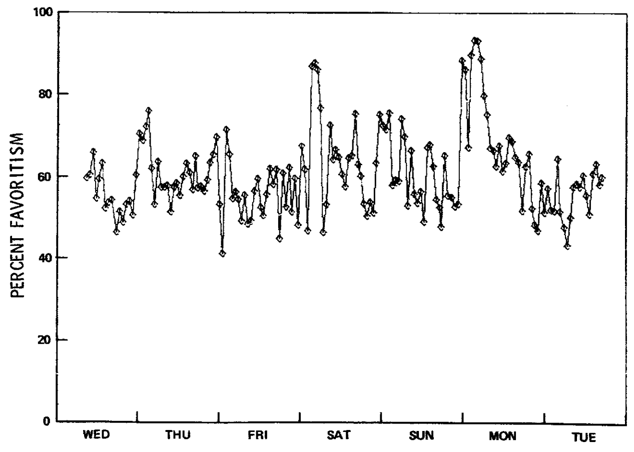


Figure 10—Percent of traffic to most favored destinations

* This, and the other "hourly" plots show points separated by approximately 56 minutes (an integral multiple of the accumulated statistics interval of roughly 7 minutes). The separation between the days on the horizontal axes occurs at midnight.

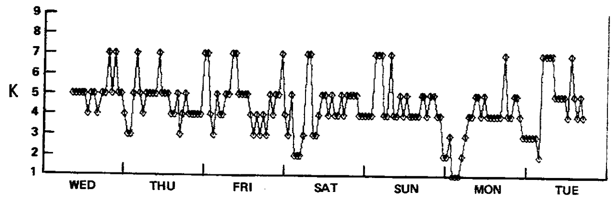


Figure 11—Number of favored destinations required to achieve 90 percent traffic

A further illustration of the non-uniformity of the traffic is seen in Figure 7. Here, we have plotted the cumulative percent of messages sent from the n busiest sources. Notice that over 80 percent of the traffic is generated by the busiest one third of the sites. A similar effect is true for the busiest (most popular) destinations. Even more striking is Figure 8, in which we have plotted the cumulative percent of traffic between site-pairs. Notice that 90 percent of the total traffic is between 192 (12.6 percent) of the site-pairs.

The interesting property of favoritism is shown in Figure 9. For each source, the destinations may be ordered by the frequency of messages to those destinations. In Figure 9, we show (summed over all sources) the percent of traffic to a source's n most favored destinations. If these orderings and percentages remained invariant over time (i.e., a stationary traffic matrix), then one could use this information in the topological design; however, it can be shown[4,13] that both the network design and performance are relatively insensitive to changes in the traffic matrix (and so, a uniform requirement is usually assumed). Note that 44 percent of the network traffic goes to the most favored sites! (A uniform traffic matrix would give a percentage of only $1/N = 2.56$ percent). Also,

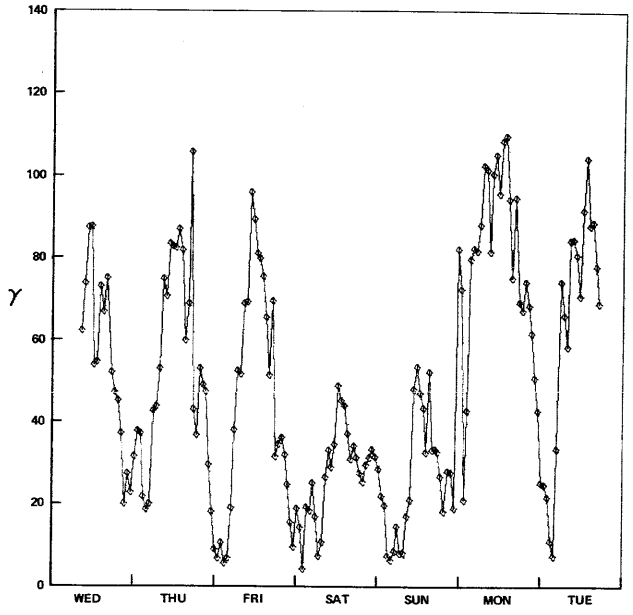


Figure 12—Arrival rate of HOST messages per second (γ)

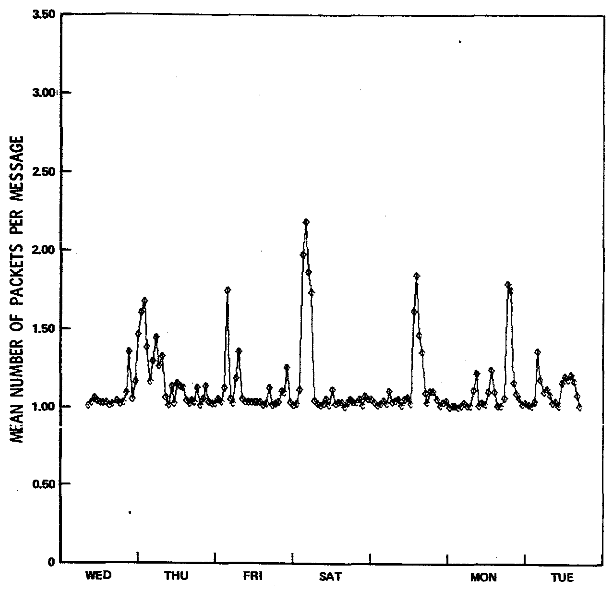


Figure 13—Mean number of packets per message

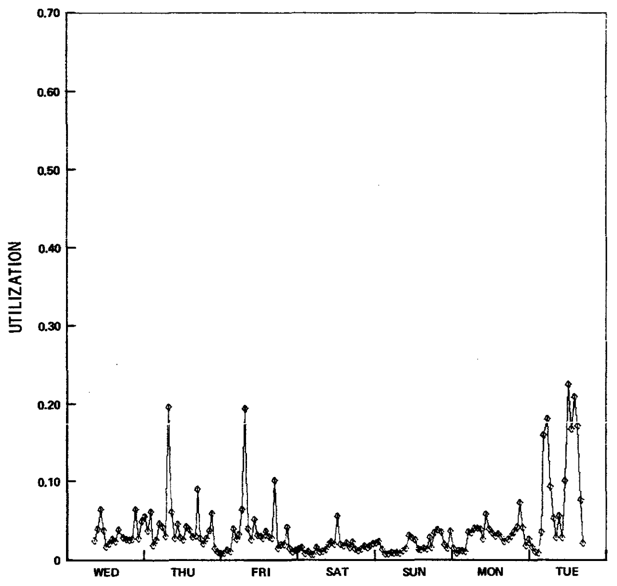


Figure 15A—Utilization of most heavily used channel in each hour (without overhead)

90 percent of the traffic goes to the nine most favorite sites; however, it is important to realize that this involves more than nine sites (in fact, 33 unique destinations are involved), since each source need not have the same set of nine most favorites. This favorite site effect is more dramatically displayed in Figure 10, which shows the percent of traffic to the most favored destination of all sources on an hourly basis. Most of the traffic (a minimum of 40 percent and an average of 61 percent) was caused by conversations between the N sources and their favorites. There are N^2 pairs in total; thus, on a weekly basis, the N favorites account for $.44N$ times the traffic they would have generated if the traffix matrix had been uniform (on an hourly basis it is $.61N$). Note that the favorite site effect must increase as we shrink the time interval over which "favorite" is defined;* in fact, if we choose an interval comparable to a message transmission time, then the

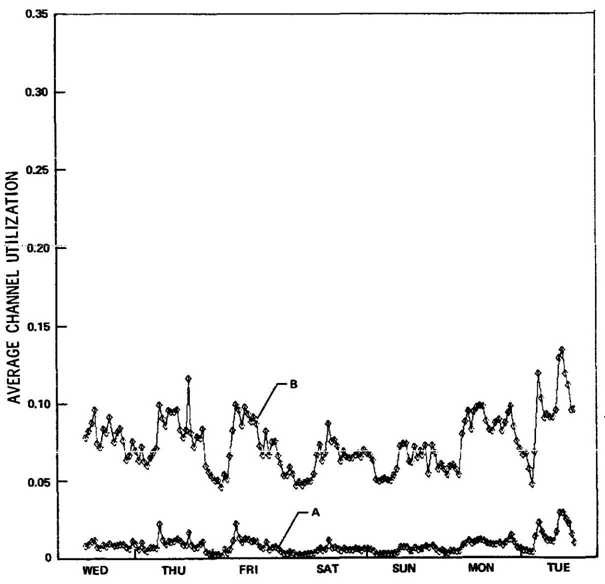


Figure 14—Network-wide mean channel utilization: (A) without overhead; (B) with overhead

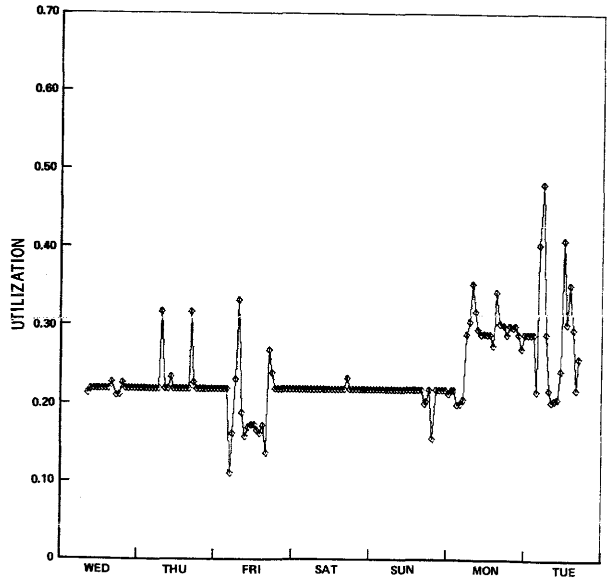


Figure 15B—Utilization of the most heavily used channel in each hour (with overhead)

* We are pleased to acknowledge the assistance of Stanley Lieberson in explaining this effect.

most favorite sites will account for almost 100 percent of the traffic, since the name of each source's favorite site will change dynamically to equal the name of the destination site for this source's traffic of the moment. Thus, the amount of traffic due to favorite sites has an interpretation which changes as the time interval changes. The weekly value of 44 percent has two possible interpretations. The first is that there exists a true phenomenon of favoritism due, perhaps, to the existence of a few useful "server" systems. The second interpretation is that network users are lazy; once a user becomes familiar with some destination HOST, he continues to favor (and possibly encourages others to favor) that HOST in the future rather than experimenting with other systems, too. A further explanation for this phenomen is that it is not especially easy to use a foreign HOST at this stage of network development; this trend should diminish as network use becomes more user oriented.

Related to Figure 10 is Figure 11 in which we have plotted the number, K, of favored destinations necessary to sum to 90 percent of the overall traffic on an hourly basis. This means that in any hour, 90 percent of the messages were sent between at most NK of the total $N^2 = 1521$ pairs in the network. Notice that K has a maximum hourly value of 7 (this is less than the weekly average of $K = 9$ due to the smaller averaging interval as discussed above). Therefore, for any hour, it requires at most 18 percent of the site-pairs to send 90 percent of the messages (in the most extreme case, $K = 1$ and so for those hours at most $1/N$ or 2.56 percent of the site-pairs sent 90 percent of the messages).

Let us now discuss other global measures of the network behavior. In Figure 12, we show the average rate at which HOST messages were generated (per second) on an hourly basis; this gives us an indication as to when the work was done on the network. There are no real surprises here: the curve shows a predominance of traffic during daylight hours and on weekdays. It is interesting that Monday had noticeably heavier traffic than the other weekdays (were the users manifesting feelings of guilt or anxiety for having slowed down during the weekend?). Observe that a truly worldwide network with its time zones could perhaps take advantage of these hourly and daily slow periods.

Figure 13 illustrates the change in network use as a function of time by showing the time behavior of the mean number of packets per message. The peaks are associated with those hours during which file transfers dominated the interactive traffic. These peaks in general occur during off-shift hours (as with incest). Perhaps users feel that they get better data rates, reliability, or HOST service late at night; or, perhaps the background of file transfers is continually present, but is noticed only when the interactive users are alseep.

The internal traffic on channels is one measure of the effectiveness of the network design and use. In Figure 14, we show the channel utilization averaged over the entire network on an hourly basis, both with and without overhead. The utilization (whose weekly average was .071 if overhead is included or .0077 neglecting overhead) is rather low and suggests that the lines in the network have a great deal of excess capacity on the average (this excess capacity is desirable for peak loads). The maximum hourly line load (including overhead, and averaged over all channels) was approximately 13.4 percent (occurring five hours before the end of the measurement) and corresponded to an internal network flow of roughly 600 KBPS; without overhead the maximum hourly average utilization was approximately 2.9 percent (129 KBPS internal traffic). It is interesting to obesrve the ***heaviest loaded line*** during each hour; this we plot in Figure 15

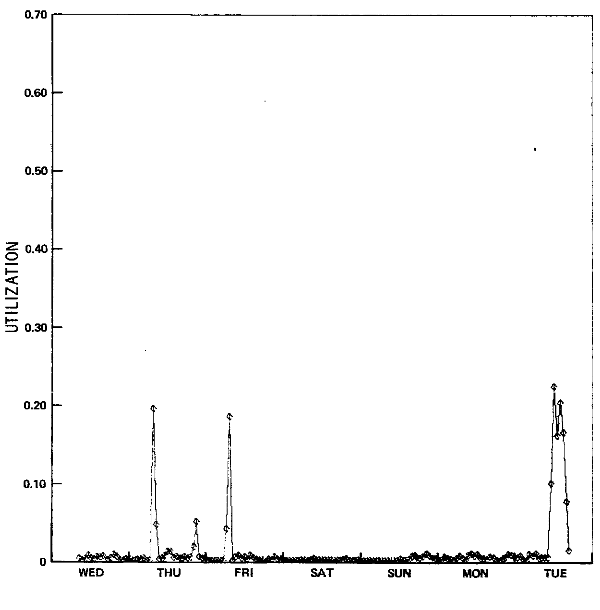


Figure 16A—Utilization of the channel (GWCT to CASE) with the highest hourly average (without overhead)

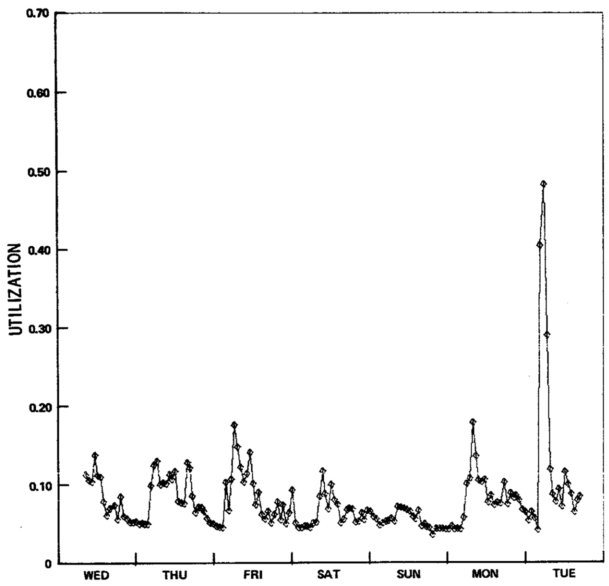


Figure 16B—Utilization of the channel (HARV to ARBD) with the highest hourly average (with overhead)

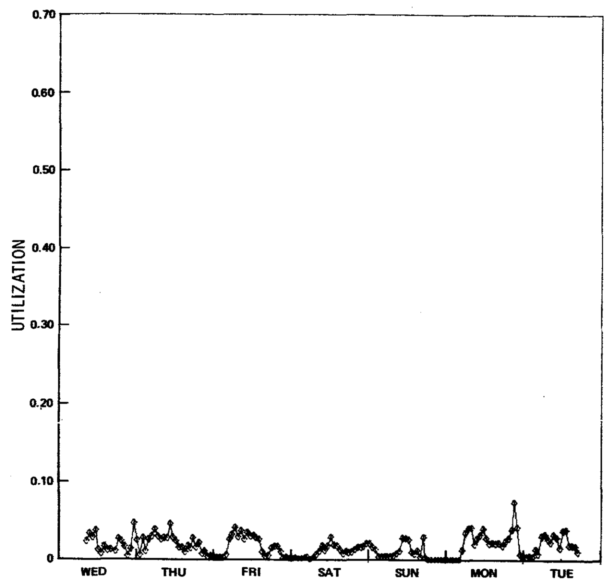


Figure 17A—Utilization of the channel (ISI to RMLT) with the highest weekly average (without overhead)

both without (part A) and with (part B) overhead. Note from part B that the *busiest line of any hour* (HARV to ABRD) had a utilization of 0.48 for that hour; without overhead the busiest line (GWCT to CASE) had a utilization of 0.225 for its busiest hour. Over the seven days, these channels had hourly load histories as shown in Figure 16. Note how bursty the traffic was on these lines (even averaged over an hour). Another interesting line is that one which had the maximum load averaged over the week. Neglecting routing

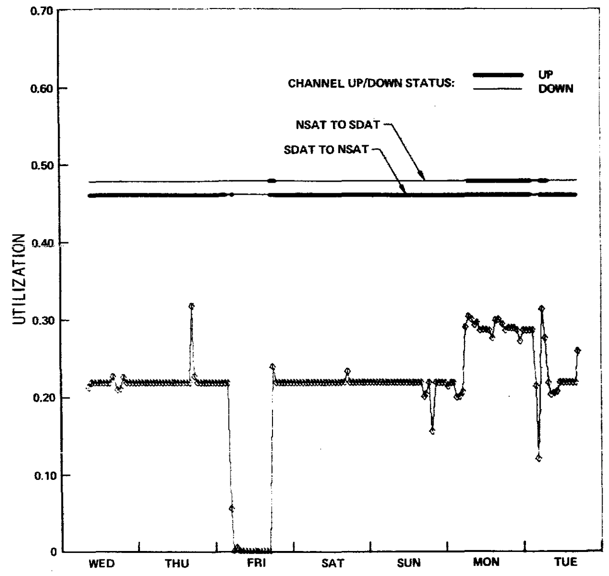


Figure 17B—Utilization of the channel (SDAT to NSAT) with the highest weekly average (with overhead)

updates and all other overhead the channel from ISI to RMLT had the largest weekly load (0.017), and its hourly behavior is shown in Figure 17A; again we see bursty behavior. If we include overhead then the satellite channel to Norway (SDAT to NSAT) had the *largest utilization averaged over the week* since it is only a 7.2 KBPS channel and therefore, all traffic placed almost seven (50/7.2) times the load on it (in this case, roughly 2KBPS, or 28 percent of the line, is used for routing updates alone). The hourly history for this channel is shown in Figure 17B. Also on this figure we have shown the UP/DOWN status of this line (in both directions).* Note that the channel was operational in both directions for a small fraction of the measurement (mainly on Monday) and only during this time was it carrying its own routing updates as well as responses to the NSAT to SDAT channel's routing updates in the form of "I heard you's"; this gives the 28 percent overhead mentioned above. This channel was down for a large part of Friday during which time it carried no traffic. For the rest (most) of the week the NSAT to SDAT channel was down and so no "I heard you" traffic was recorded on the SDAT to NSAT channel as can be seen in Figure 17B.

With few exceptions the channels in the network are fairly reliable. Over half of the channels reported packet error rates less than one in 100,000. The average packet error rate was one error in 12,880 packets transmitted. Of the 86 channels in the network 14 reported no errors during the seven days,

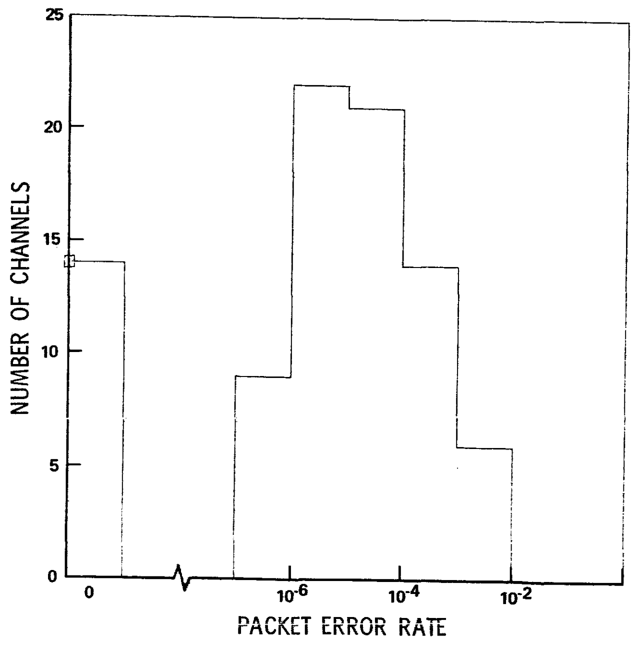


Figure 18—Channel packet error behavior

* Our measurements actually give the UP/DOWN status of the IMPs as seen by the NMC. When NSAT is declared down, we have displayed the NSAT to SDAT channel as being down in Figure 17B, and similarly, when SDAT is declared down we have shown the SDAT to NSAT (and the NSAT to SDAT) channel down.

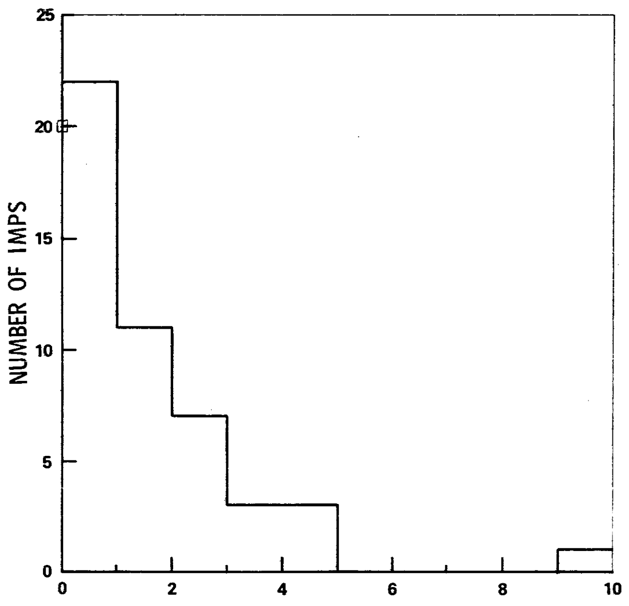


Figure 19—IMP failure behavior

while six channels had packet error rates worse than one in 1000. The worst case was one in 340 packets for the channel from RADT to LL. While these error rates are large enough to warrant the inclusion of error detection hardware and software, they are small enough so that traffic flow through the network is not impaired. In Figure 18, we show the error behavior of these lines during the seven day measurement. The failure rate of the IMPs should be included here, but clearly the seven day measurement is insufficient for this purpose. For completeness, therefore, in Figure 19 we show the performance characteristics of the IMPs over a 19 month interval (June 1972 through December 1973).[14] The average IMP down rate was 1.64 percent, with the worst case being 9.13 percent.

MODEL FOR NETWORK DELAY

In this section, we present a network delay model originally introduced by Kleinrock[12] and which was extended by Fultz[15] and Cole.[11] We then further extend this model to fit the specific implementation of the ARPA network. Following the model formulation, we present a comparison between the predicted and measured delay.

With the assumption of negligible nodal processing delays and channel propagation delays, the average message delay T (the time to traverse the network from source to destination) originally appeared as[12]

$$T=\sum_{i=1}^{M}\left[\frac{\lambda_i}{\gamma}T_i\right]$$

where

λ_i = the mean arrival rate of messages to the ith channel,
γ = the mean arrival rate of messages entering the network,
T_i = the mean time spent waiting for and using the ith channel, and
M = the number of channels in the network.

This very general result is easily extended to include nodal and propagation delays as follows:

$$T=K+\sum_{i=1}^{M}\left[\frac{\lambda_i}{\gamma}\left(\frac{1}{\mu C_i}+P_i+K+W_i\right)\right]$$

where

$1/\mu$ = mean message size,
C_i = capacity of the ith channel,
P_i = propagation delay on the ith channel,
K = nodal processing delay, and
$W_i=T_i-1/\mu C_i$ = waiting time in queue for channel i.

The delay analysis now simply requires that we solve for W_i. Perhaps the simplest (Markovian) assumption is[16]

$$W_i=\frac{\lambda_i/(\mu C_i)}{\mu C_i-\lambda_i}$$

When the queueing delay due to control traffic is also considered, we have

$$W_i'=\frac{\lambda_i'/(\mu' C_i)}{\mu' C_i-\lambda_i'}$$

where

λ_i' = arrival rate of data messages and control messages to the ith channel, and
$1/\mu'$ = mean message size including control messages.

Removing the assumption that nodal processing delay is constant and including the destination HOST transmission time we obtain the following expression for the average delay experienced by single packet messages.

$$T_{\mathrm{SP}}=\sum_{i=1}^{M}\left[\frac{\lambda_i}{\gamma}\left(\frac{1}{\mu C_i}+P_i+K_l+\frac{\lambda_i'/(\mu' C_i)}{\mu' C_i-\lambda_i'}\right)\right]+\sum_{i=1}^{N}\left[\frac{\gamma_{\cdot j}}{\gamma}\left(K_j+\frac{1}{\mu_{\mathrm{H}}C_{\mathrm{H}j}}\right)\right]$$

where

K_l is the packet processing time at node l (l is the origin node of channel i),
$\gamma_{\cdot j}$ = the mean departure rate of messages from the network to the HOSTs at site j,
$1/\mu_{\mathrm{H}}C_{\mathrm{H}j}$ = the mean transmission time of messages to a HOST at site j, and
N = the number of nodes in the network.

The above formulae assume unpacketed message traffic, while in the ARPANET, messages are divided into from 1 to

8 packets. Fultz[15] and Cole,[11] therefore, extended the model to obtain the mean delay experienced by multi-packet messages

$$T_{MP}=K+\sum_{i=1}^{M}\left[\frac{\lambda_i}{\gamma}\left(\frac{1}{\mu C}+P_i+K+W_i'\right)\right]$$
$$+(\bar{m}-1)\left(\frac{1}{\mu C}+\sum_{jk}\left[\frac{\gamma_{jk}}{\gamma}\bar{\tau}_{jk}\right]\right)$$

where

C = line capacity (temporarily assumed constant)
$\bar{m}$ = mean number of packets in a multipacket message,
γ_{jk} = the arrival rate of messages from j to k, and
$\bar{\tau}_{jk}$ = mean inter-packet gap time for messages from source j to destination k.

It is difficult to measure $\bar{\tau}_{jk}$ for each j,k pair in the network. We, therefore, introduce an approximation due to Cole,[11] which yields

$$E[\tau(n\text{ hops})]=\frac{\rho(1-\rho^{(n-1)})}{1-\rho}\,\frac{1}{\mu C}$$

The above expression gives the expected value of τ_{jk} for nodes j and k which are n hops apart. It assumes that the channel utilizations ρ_i for the channels in the path from j to k are constant and equal to ρ. The path is assumed unique and the channel capacities are constant with value C. We will use the first two assumptions to obtain an approximation to the network-wide mean interpacket gap. Note that the average path length traveled by a message is given by

$$\bar{n}=\frac{\lambda}{\gamma}$$

where

$$\lambda=\sum_{i=1}^{M}\lambda_i$$

The average line utilization is

$$\bar{\rho}=\frac{\displaystyle\sum_{i=1}^{M}\frac{\lambda_i'}{\mu' C_i}}{M}$$

Where once again we let C_i = capacity of the ith channel. The time it takes to transmit a full packet averaged over all channels in the network is

$$\bar{S}_F=\sum_{i=1}^{M}\left[\frac{\lambda_i}{\lambda}\,\frac{1}{\mu_F C_i}\right]$$

where $1/\mu_F$ = the length of a full packet.
Thus, we will use the following approximation for $\bar{\tau}$:

$$\bar{\tau}=\frac{\bar{\rho}(1-\bar{\rho}^{(\bar{n}-1)})}{1-\bar{\rho}}\bar{S}_F$$

Removing the assumptions of constant K and C, adding the HOST transmission time, and assuming that the last packets of multipacket messages have the same mean length as the single packet messages, we have the average message delay for multipacket messages:

$$T_{MP}=\sum_{i=1}^{M}\left[\frac{\lambda_i}{\gamma}\left(\frac{1}{\mu_F C_i}+P_i+K_l+\frac{\lambda_i'/(\mu' C_i)}{\mu' C_i-\lambda_i'}\right)\right]$$
$$+\sum_{i=1}^{M}\left[\frac{\lambda_i}{\lambda}\left((\bar{m}-2)\frac{1}{\mu_F C_i}+\frac{1}{\mu C_i}\right)\right]$$
$$+\sum_{j=1}^{N}\left[\frac{\gamma_{\cdot j}}{\gamma}\left(K_j+\frac{1}{\mu_{FH} C_{Hj}}\right)\right]$$
$$+(\bar{m}-1)\bar{\tau}$$

where $1/\mu_{FH}C_{Hj}$ = the transmission time of a full packet to a HOST at site j.

Let β be the fraction of the total number of messages which are single packet messages. We obtain the final expression for average message delay (from source to destination) in the network.

$$T=\beta T_{SP}+(1-\beta)T_{MP}$$

The measure of delay which is supplied by the IMPs is round-trip delay. Therefore, in order to compare the model with the measurements we need an expression for round-trip delay (i.e., we need to include the average RFNM delay T_{RFNM} in the model). A RFNM is simply another single packet message traveling from destination to source. Thus, it experiences the single packet message delay T_{SP} with an appropriate value for μ and λ without the HOST transmission term as follows:

$$T_{RFNM}=\sum_{i=1}^{M}\left[\frac{\lambda_{Ri}}{\gamma}\left(\frac{1}{\mu_R C_i}+P_i+K_l+\frac{\lambda_i'/(\mu' C_i)}{\mu' C_i-\lambda_i'}\right)\right]$$
$$+\sum_{j=1}^{N}\left[\frac{\gamma_{j\cdot}}{\gamma}K_j\right]$$

where

λ_{Ri} = the mean arrival rate of RFNMs to channel i,
$1/\mu_R$ = the length of a RFNM, and
$\gamma_{j\cdot}$ = the mean departure rate of RFNMs from the network to the HOSTs at site j (= the mean arrival rate of messages from the HOSTs at site j to the network)

The expression for mean round-trip delay T_R is therefore,

$$T_R=T+T_{RFNM}$$

For the week-long measurement we calculated the zero-load value of T_R and obtained $T_R=69$ msec; the hourly variation of this quantity is shown in Figure 20. The source of the variation is the shift in the origin-destination traffic mix. This zero-load case corresponds to forcing λ_i and γ to zero, (keeping the same ratio as before for each i). The zero load value must be less than the measured value, and compares with the measurements displayed in Figure 21. This emphasizes the fact that the network is introducing very small congestion

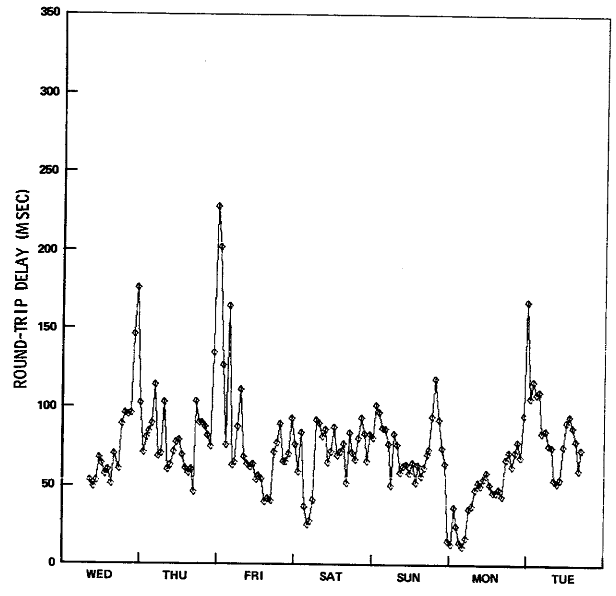


Figure 20—Computed (zero load) average message delay

effects. Furthermore, in Figure 22 we show the hourly variation of T_R (whose weekly average was $T_R=73$ msec) calculated for the actual load value as measured.

The model presented above is rather complex due mainly to the fact that not all channels (or IMPs) need have the same speed. In addition, the waiting time terms complicate the expressions as well, and represent the part of the model which is most subject to question (i.e., the Markovian assumptions). However, from Figures 20 and 22, we see that the zero-load and measured load calculations are nearly the same. This shows that the effect of W_i' is quite negligible and so any improvement over Markovian assumptions will yield negligible changes to T_R. This suggests a far simpler no-load model for estimating T_R as follows.[17] The expressions for T_{SP} (and T_{RFNM} which is similar in form), may be simplified by dropping the W_i' terms, and setting all $K_i=K$ (a constant), all $C_i=C$ (a constant at 50 KBPS), and $C_{Hj}=C_H$ (a constant at 100 KBPS). The result is

$$\hat{T}_{SP}=\bar{n}\left(\frac{1}{\mu C}+K\right)+K+\frac{1}{\mu_H C_H}+\sum_{i=1}^{M}\left[\frac{\lambda_i}{\gamma}P_i\right]$$

(and a similar expression for $\hat{T}_{RFNM}$). Except for the last summation, these parameters are easily computed. For the sum, one must estimate (or measure) the channel traffic λ_i and the network throughput γ. The propagation delays P_i are known constants. With these simplifications (and assuming $\beta=1$, since the measured value of $\beta=0.96$ was observed), we then have the approximation

$$\hat{T}_R=\hat{T}_{SP}+\hat{T}_{RFNM}$$

Our calculation gives $\hat{T}_R=70$ msec* which is an excellent approximation to the earlier stated value of $T_R=69$ msec (at zero-load) and $T_R=73$ (at measured load)!

On the other hand, the measured value of $T_R=93$ msec is significantly larger than measured load estimate of the model of $T_R=73$ msec. This difference is due to the fact the model does not include: any delay by the destination HOST in accepting the message; any delay due to the request for storage at the destination IMP; exact data on P_i; time variations in

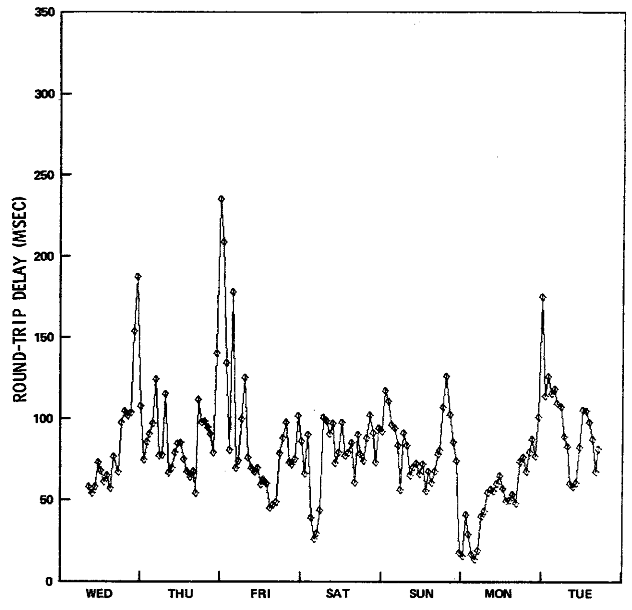


Figure 22—Computed (measured load) average message delay

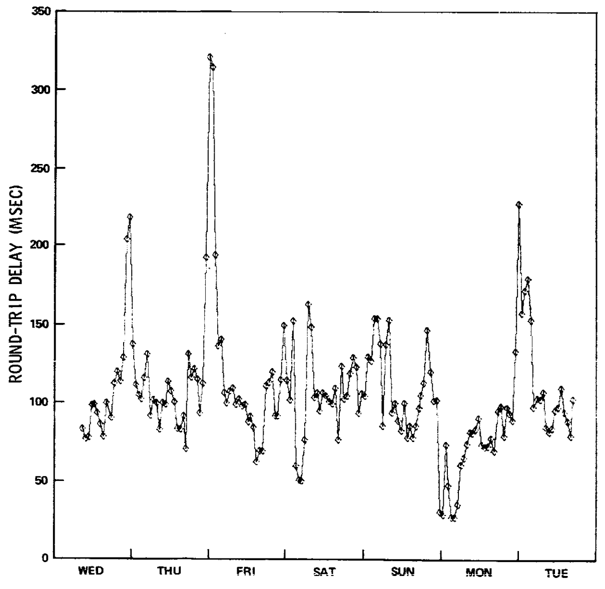


Figure 21—Measured average message delay

* The components for $\hat{T}_R$ are: $\bar{n}=3.31$, $1/\mu C=8.2$ msec, $K=.75$ msec, $1/\mu_H C_H=2.75$ msec, the propagation sum $=11.4$ msec, and $1/\mu_R C=3.36$ msec.

ρ finer than the hourly computations used; and non-Markovian assumptions. All the above omissions (except possibly the last) will increase the computed value of T_R.

CONCLUSIONS

The purpose of this paper was to present results of a week-long measurement of the ARPANET traffic behavior. In reporting upon the results of that experiment, we have observed a number of quantitative relationships which suggest that values assigned to certain of the network parameters should perhaps be reexamined. For example, we observed that the vast majority of messages are single-packet messages and one wonders at the wisdom of providing within the network the rather sophisticated mechanisms for handling multi-packet messages. Furthermore, we observed that the single-packet messages themselves are extremely small and it may be possible to improve the efficiency of the network if, in fact, the maximum packet lengths (and, therefore, the IMP buffer length) were reduced; one source of these small packet lengths is the preponderance of interactive traffic which typically creates packets containing one or a few characters. The mode of communication perhaps itself needs to be reexamined in an attempt to improve the network efficiency while maintaining a comfortable interactive feeling and response time. Incest is rampant in the network and it might be worthwhile to investigate other means for handling such traffic. Favoritism is (and perhaps will remain) even more dominant and how one would take advantage of this effect is not at this point clear. The non-uniformity of the traffic is striking and future network designs should attempt to capitalize upon this feature. The time variation of network use was discussed above and we see a fairly cyclic behavior both in traffic intensity and type of use. The lines themselves are not heavily utilized, and at the same time the network delays are so small as to render the network invisible to the typical user. We have described, in addition, a fairly extensive model for network delay and comparing it to our measured results it seems to be a fairly valid model both for single-packet and multi-packet messages. We also give a simplified model which appears adequate.

In this paper, our major purpose has been to report the measured results from our experiment. Secondarily, we have scratched the surface in attempting to evaluate and draw conclusions regarding the chosen values for some of the design parameters. In this effort, we have avoided the depth of discussion required to make a meaningful evaluation of these parameters, but rather have discussed their values only in terms of the measured data. For example, the choice of IMP buffer size depends upon many considerations beyond those we have measured (e.g., IMP processing speed, interrupt structure, line error rates, maximum network throughput, etc.); therefore, the presentation and commentary on the measured data given herein should certainly not be used alone in the selection of network parameters. The broad class of issues which must be included in decisions of this type are discussed, for example, in Kahn.[18]

The experiment described above is repeated every two months at the Network Measurement Center, and has so far produced results similar in flavor to those reported upon here. Numerous other experiments are currently being conducted and many more are in the planning stages. It is only through such experiments and through careful evaluation of the measured data that one can gain understanding of the network behavior, which in turn impacts the design and growth of the network.

ACKNOWLEDGMENT

We gratefully acknowledge the contributions of Holger Opderbeck, Stanley Lieberson, and the remainder of the staff at UCLA-NMC.

REFERENCES

1. Roberts, L. G., and B. D. Wessler, "Computer network development to achieve resource sharing," *AFIPS Conference Proceedings*, 36, pp. 543-549, SJCC, Atlantic City, N.J., 1970.
2. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The interface message processor for the ARPA computer network," *AFIPS Conference Proceedings*, 36, pp. 551-567, SJCC, Atlantic City, N.J., 1970.
3. Kleinrock, L., "Analytic and simulation methods in computer network design," *AFIPS Conference Proceedings*, 36, pp. 569-579, SJCC, Atlantic City, N.J., 1970.
4. Frank, H., I. T. Frisch, and W. Chou, "Topological considerations in the design of the ARPA computer network," *AFIPS Conference Proceedings*, 36, pp. 581-587, SJCC, Atlantic City, N.J., 1970.
5. McKenzie, A.A., *HOST/HOST Protocol for the ARPA Network* ARPA Network Information Center #8246, January 1972.
6. Ornstein, S. M., F. E. Heart, W. R. Crowther, H. K. Rising, S. B. Russell, and A. Michel, "The Terminal IMP for the ARPA network," *AFIPS Conference Proceedings*, 40, pp. 243-254, SJCC, Atlantic City, N.J., 1972.
7. Frank, H., R. E. Kahn, and L. Kleinrock, "Computer communication network design—Experience with theory and practice," *AFIPS Conference Proceedings*, 40, pp. 255-270, SJCC, Atlantic City, N.J., 1972.
8. Crocker, S. D., J. F. Heafner, R. M. Metcalfe, and J. B. Postel, "Function-oriented protocols for the ARPA Computer Network," *AFIPS Conference Proceedings*, 40, pp. 271-279, SJCC, Atlantic City, N.J., 1972.
9. *Specifications for the Interconnection of a HOST and an IMP*, Bolt, Beranek and Newman, Inc., Cambridge, Mass., Report No. 1822 May 1969.
10. McKenzie, A. A., B. P. Cosell, J. M. McQuillan, and M. J. Thrope, "The Network Control Center for the ARPA Network," *Proceedings of the First International Conference on Computer Communication*, 1, pp. 185-191, Washington, D. C., October, 1972.
11. Cole, G. D., *Computer Network Measurements: Techniques and Experiments*, Engineering Report No. UCLA-ENG-7165, University of California, Los Angeles, Calif., 1971.
12. Kleinrock, L., *Communication Nets: Stochastic Message Flow and Delay*, McGraw Hill, N.Y., 1964, reprinted by Dover, N.Y., 1972.
13. Gerla, M., *The Design of Store-and-Forward (S/F) Networks for Computer Communications*, Engineering Report No. UCLA-ENG-7319, University of California, Los Angeles, Calif., 1973.
14. McKenzie, A. A., Letter to S. D. Crocker, 16 January 1974.
15. Fultz, G. L., *Adaptive Routing Techniques for Message Switching Computer-Communication Networks*, Engineering Report No. UCLA-ENG-7252, University of California, Los Angeles, Calif., 1972.
16. Kleinrock, L., *Queueing Systems, Volume I: Theory*, Wiley, N.Y., 1974.
17. Kleinrock, L., *Queueing Systems, Volume II: Computer Applications*, Wiley, N.Y., 1974.
18. Kahn, R. E., "Resource-Sharing Computer Communication Networks," *Proceedings of the IEEE*, 60, pp. 1397-1407, November 1972.

The potential role of the computer in intuition and self development

by DAVID V. TIEDEMAN

Northern Illinois University
De Kalb, Illinois

ORIENTATION

From 1966-69, several colleagues and I had opportunity to design and implement a prototypical ISVD (Information System for Vocational Decisions). Dr. JoAnn Harris[1] has been good enough to record that the ISVD and several sister computer guidance systems forecast and piloted the following advances in uniting the capabilities of computers and the purposes of guidance:

1. "Increased use of visuals to supplement words.
2. "Development of programs which will allow counselees to simulate vocational, educational, and personal-social experiences.

⋮

4. "Development of languages which allow the student to respond to the computer in his own language.
5. "Development of programs which will allow the counselee to prepare his own instructions to the computer and thus alter the pre-designed processes." (page 10)

Of the five items Dr. Harris listed as harbingers of computer involvement in guidance the only one which ISVD lacked was the third one, group interaction via computer terminals. The ISVD made no computer provision for that function. The ISVD planned that that function be discharged in its system without computer involvement.

Today I am without my guidance computer. Neither the computer technology nor the educational economics of these days are up to the dream which I had for the ISVD.

Dr. Martin Katz and Dr. Harris are two eminent colleagues in the development of effective unions of computer capability and guidance purposes who entered the field about when I did but have the good fortune still to be in business since they gauged technological capability and user purse more level headedly than I did. Nevertheless we have all continued to remain interested in the common problem of convincing the public that they need the adjuvant computer capability in guidance now available to them. My colleagues have therefore been good enough to invite me to participate with them in this symposium which in its entirety will reveal the effective role which the computer has come to play in career guidance.

The symposium is divided into four parts. I take my own function to be merely that of revealing the potential role of the computer in intuition and self development. Dr. Katz and Dr. Harris will then describe two operational computer guidance examples, SIGI (System for Interactive Guidance and Information) and CVIS (Computerized Vocational Information System) respectively. Mr. Larry Blasch concludes the symposium by considering computer-based career guidance systems of the future.

ERIC AND THE INFORMING PROCESS

Risk is an essential condition for solving the paradox of intuition and development. Possibilities and progress must always be traded off: progress is bought at the expense of possibilities; possibilities at the expense of progress. The person comes to know this fact at increasingly more fundamental levels of understanding as he develops himself hierarchically, the way in which I hypothesize that self development takes place.[2,3] As the person also comes to know the truth in Polanyi's[4] assumption that "... we can know more than we can tell" (page 4), he comes to know at increasingly more fundamental levels of his being that his own intuition is the engine of his intelligence. The more a person risks to his intuition, the more he develops hierarchically.

My understanding of the gaining of confidence in intuition through risk of the self evolved is my struggle to fashion a relationship between a person and a computer which has the purpose of self enhancement, not self contraction. I offer an account of that struggle in this paper in expectation that its revelation may harmonize with understandings you also have about the potential role of the computer in intuition and self development.

ERIC (Educational Resources Information Center)

Let's start with ERIC, the Educational Resources Information Center. ERIC acquires, selects, abstracts, and indexes documents. Abstracts and index terms of the selected documents are transferred to computer tapes. The selected documents themselves are transferred to microfiche provided that their authors have authorized such copying if the material is copyrighted. Journal articles receive similar treatment except that a journal article is only annotated, not abstracted, and journal articles are not put on microfiche.

The computer tapes which contain citations of educational research and practice and associated index terms and abstracts or annotations in each case constitute a new adjuvant power for man's mind, a power which I believe is of great import. Although the system has been in existence only since the mid 60's, the educational resources incorporated into ERIC since then are now fairly complete and quite accurately assembled in one place. Completeness and accuracy constitute two of the conditions which man attempts to approximate ever more closely with his mind. By letting their minds cooperate with ERIC, users now prove able to achieve both conditions more fully in relation to educational resources assembled since the mid 60's. The availability of access to these resources through computer tapes gives a flexibility to the use of this more complete and more accurate store than was available prior to ERIC. All that the user has to do is let ERIC become adjuvant to his mind. ERIC and its microfiche then open for every person new vistas of contact with educational developments in the United States and other parts of the world heretofore unavailable.

The adjuvant relationship with ERIC is made available to informed inquirers on the general basis noted above. ERIC also engages in educating potential users so they can take advantage of ERIC's adjuvant capability as well.

In order to keep a general level of contact with ERIC available to all, ERIC publishes *Research In Education* and *Current Index to Journals in Education* on a monthly basis. Both of these journals essentially list currently accessioned material by main entry, major subjects and author. The main entry includes the abstract or annotation, as the case may be.

Sixteen Clearinghouses are an essential current feature of the ERIC network. Clearinghouses cast their nets to acquire documents and assume responsibility for selection, abstracting, and indexing of documents and journal articles. Clearinghouses are also responsible for information analyses in each of their scopes. These information analyses synthesize the literature around current issues or problems of popular concern. Clearinghouses also provide individual service to users upon request provided that such service does not constitute an inordinate drain on time which must be husbanded for the essential accessioning tasks.

ERIC as an information system

Several years ago, Walz and Rich[5] who both had responsibilities in the ERIC Clearinghouse for Counseling and Personnel Services caught my imagination with their article, "The Impact of Information Systems on Counselor Preparation and Practice." After introducing ERIC on somewhat the same terms I have used above, Walz and Rich first listed the essential characteristics of an information system as follows:

1. "The major objective of any information system is to organize and store information in a form which maximizes the user's ability to rapidly locate information which is relevant to his specific professional concerns.
2. "The major processing procedure which preserves the capability of identifying the nature of these educational materials at a later time is indexing.
3. "The primary determinate of which index terms will be used is the article which is being indexed.
4. "The outcome of indexing is a list of index terms which have been selected because they are an accurate representation of the conceptual contents of the article.
5. "The ERIC system, then, is collecting a large number of educational materials and indexing these materials for storage and subsequent retrieval.... This file may be searched using two search strategies. First a search may be conducted to identify all materials which deal with any one concept.... A second search strategy is to coordinate index terms so that the information retrieved is smaller in quantity but more directly relevant to the user's question.

Walz and Rich went on to argue that an information system which organized and stored information so that the user can retrieve information pertinent to his interest and needs will have the following predictable outcomes:

1. Use will come to be focused on synthesis and evaluation.
2. Use will reveal gaps in the information structure.
3. The use of impersonal information sources will increase because information access will be more commonly available.
4. Opportunities for interprofessional interaction will increase because the material of one profession will be indexed to the different primary terms of another profession.
5. Information rather than books will become the primary focus for retrieval. ERIC will become the user's "book."
6. Information dissemination will become broader and faster.

Finally, Walz and Rich went on to propose that the education of counselors would shift as follows because of the new availability of the ERIC information system:

1. Inquiry will become the primary focus of learning.
2. A need will arise for new learning approach skills emphasizing retrieval and the investment of facts with personal meaning.
3. Evaluation and integration will become more personally essential.

4. The pedagogy of counselor education will shift to the personal absorption of information once the information system has begun to achieve coagulation of information as it ordinarily does.
5. Counselor educators will adopt new methods of professional communication which will center on ERIC, not books and journals.
6. Counselor educators will increase their collaborative efforts.
7. The number of small esoteric information systems will increase.

ERIC and the informing process

The Walz and Rich article came to my attention shortly after several colleagues and I had begun to design the ISVD. As I will show later, the ISVD takes the inquirer himself into consideration thus facilitating his ability to turn data retrieval into information generation. I first proposed[6] this relationship in an article in the *CAPS Capsule,* the information arm of the ERIC Clearinghouse for Counseling and Personnel Services. I later went on to describe the necessary mediation process more fully in an article with Robert O'Hara.[7]

From media to mediation

O'Hara and I argued that mediation, not media, must become the central focus for education, general and vocational. The turning of occupational facts/data* into information is a personal and educational process. Therefore, the important question is the means by which media actually prove to mediate the personal education process. I elect to address in this paper the most important of our questions in vocational guidance: How may we better the personal educational process associated with vocational development?

Several years ago I had a confirming experience with a recent book by McLuhan and Fiore.[8] The younger of my sons left his copy of this book on a table in my living room for about a week, but I was not particularly interested in it because, during my early glances at it, I kept reading its title as *The Medium Is the Message.* That seemed a clever, though not an intriguing, title. One day it suddenly came to me that the book's title is *The Medium Is the Massage.* My double-take and that realization connected then with my realization that this was the process of mediation which I develop in this subsection. Thus, a hurried turn to the book reinforced my recognition that, in occupational facts/data as well as in the generality of communication treated by McLuhan and Fiore, the media are not the message. In fact, media can never be the message; only the facts which media convey are the message. The media themselves only become important in message transmission when they actually mediate transmission—when they actually massage the occupational information process as persons are exposed to occupational facts/data.

* Occupational facts/data come in two conditions, fixed and modifiable. We therefore elected to adopt the cumbersome term, "facts/data," to indicate this fact throughout our paper. Occupational facts are directly recoverable without mediation except for storage and later recovery. On the other hand, occupational data must be additionally processed by the numeric and/or linguistic routines of a mediation system.

Epistemology and pedagogy in mediation*

The point of view I outline here is one which derives important aspects of its validity from being realized again and again through a wide range of personal experiences with facts and ideas. Our "frame of reference" with regard to the interplay of facts, ideas, purposes, and action represents a form of orientation which, while it cannot be specified in the abstract, serves in any given context of personal encounter to articulate immediate concerns with issues of broader relationship and relevance.[9] Therefore, in order to articulate aspects of my immediate topic within a context of issues of broader educational concern, I address two assumptions which I consider to be inconsistent with crucial principles of educational process. It is the alternatives to these principles that I strive to describe here.

The first of these assumptions pertains to the nature of knowing and the known. There are current applications of recent media developments to issues of vocational information which appear to presume that facts, data, or information consist of bits of knowledge which correspond directly to that presumed to be the real, the true, and the knowable. In brief, knowing and the known are presumed to comprise a direct, linear relationship both in the abstract and as we realize them as dimensions of particular circumstances. I shall show the serious limitations of this position as I show how the ISVD implemented our current technological resources in the service of personally determined career development.

The second of the assumptions is in an important sense subordinate to the first, for it pertains to the nature of the relationship between acts of knowing or learning and those of teaching or counseling. This assumption suggests that, on the basis of a "correspondence" theory of knowledge, we can presume to select those aspects of the known and knowable which shall be most effective in determining a subsequent course of events toward an end that we value and which, as "means" toward that end, we call "learning." In brief, the assumption is that we can determine, in advance, both goals and procedures appropriate to the educational process in its distinctive human immediacy and variability.

These assumptions, one "epistemological" and the other "pedagogical," are inconsistent with what seems to be one of the most crucial principles of my own current work, namely, that both knowledge and the process of knowing are functions of a personal and collaborative context of exploration and confirmation—a context which is itself

* I am indebted to Gordon A. Dudley for the structure and ideas of this subsection.

defined by a nexus of human purposes expressed both overtly and covertly, both tacitly and articulately. The alternative position from which I speak suggests that talk about media cannot look in one direction only. It cannot look solely toward facts, data, information in isolation from persons and processes. I hold that the reciprocal interaction between the knower and known entails a "transactional" perspective and an array of procedures more aptly denoted by the notion of mediation. The final turn of this argument is that, because of the interplay of the tacit and articulate dimensions of knowing in the personal act of learning, the experience of mediation is that of a massage. In other words, we inevitably encounter the new with a habitual tensing of our intellectual musculature, with the result that its meaning takes initial form after that which we have long known and to which we have accommodated. Only after we have worked with (and perhaps more importantly, been worked on by) a new possibility do we relax to the point of seeing more clearly that something new has indeed been going on in, as well as around, us, cf. Piaget on assimilation and accommodation as reported by Flavell.[10]

My advocated perspective is "transactional" by virtue of the implication that both processes of teaching and learning are construed as individual and collaborative acts of "sampling," from among a wide range of on-going events (both personal and environmental), those configurations of meaning and implication which best serve to differentiate means and ends, processes of imagination and structures of knowledge, and acts of discovery and principles of verification. Within this "transactional" perspective, facts and data derive their significance as exemplifications of meaningful coherence among stable dimensions of events reflecting multiple principles of order.[11] It is this transactional perspective regarding the interplay of information and imagination which brings my ideas in harmony both with current developments in discovery teaching and the "new" curricula,[12] and with the "new" self-knowledge and creative learning developments.[13] It is a point of view from which I risk inviting the student to take advantage of my capacity to learn through his ability to teach us.

INFORMING MACHINES

*ISVD: A computer-based information system for career decisions**

In prospect for computer technology

The intent of the ISVD was to place an inquirer in potentially repeated interaction with a computer-centered environment programmed for his inquiry, not just for prompt reinforcing of stimulus-response contiguity. The contexts for the inquiries were education, occupation, military service, and family living. The inquirer was permitted to elect at will among contexts. The System was constructed so as to expect the inquirer to learn how to harmonize his goals and their consequences by means of repeated inquiries in these four important realms of personal activity.

As required by the principle of mediation, the primary goal of the ISVD was inquiry, not reinforcement. Because the System put the inquirer in direct relation with his evolving history and intentions to the extent that such can be motivated and represented through the numbers, letters, and processing available in computer reckoning, it became possible to avoid one of the fears which the public has of using computers in guidance, namely, the fear that computers will determine lives by making decisions for, not with, persons. The System let any inquirer experience practically the same joy and frustration which computer devotees daily do—the realization that the answer is in a devotee of computers, not in the machine. Despite our occasional regret upon such realizations, we know that we still persevere. Therefore, the assumption of the ISVD was that any person can and will persevere through inquiry. A further assumption was that repeatedly-experienced failure to find full solutions to questions can be fashioned into mature capacity to proceed on inadequate bases in adult life as an inquirer is brought to realize the care used in fashioning a System which can take him down the path of, but never completely into, awareness of the operation of his motivational system.

ISVD was therefore different from systems now organized for computer-aided instruction or educational data processing. The ISVD subsumed those conceptions as intermediate in the condition of education for responsible career decisions. However, the primary professional task was to construct a meta-system which permitted analysis and response direction in terms of the majority of the variables of this expected responsibility.

How was that accomplished?

The career and choices in career development

The context of vocational decision-making offers excellent opportunity for realization of my intention when the computer is given centrality, but necessary incompleteness, in the interacting system in which career development emerges. O'Hara and I[14] define career as personally-given direction in developing vocational maturity. I therefore bind a career with expectation that the exercise of personal intention brings with it accountability for self-directed and corrected activity. Therefore, I expect that career development requires emergence of self-initiated and controlled activity for which a person permits himself to be held to account. When persons do so, we have opportunity to give power to the process of social control by encouraging the independence of freedom and the interdependence of social consciousness.

The forming of career involves a set of decisions which

* Other principal investigators of the Information System for Vocational Decisions were Russell Davis, Richard Durstine, Allan Ellis, Wallace Fletcher, Edward Landy, Robert O'Hara and Michael Wilson.

are made throughout life, in the joint contexts of education, vocation, military service, and family. The object, plan, and progress of decisions in each of these areas have their own characteristics which are reported in some detail in O'Hara and Tiedeman.[7]

The system

General framework

The ISVD was deliberately named despite the fact that my connotations for its words are not presently entirely a matter of common parlance. My word "Information" connotes the placing of facts/data into the context of use. This use of the word emphasizes my belief that facts/data require their context of use if they are to be conceived as information.

Students and workers were permitted to turn educational and occupational facts/data into information through the System. Thus the user became an explicit part of my connotation of "System." My connotation reflects the intention to offer the user complete responsibility in choice of educational and vocational goals. Although it is probably inevitable that the computer will be blamed for "error," I did not intend to let users of the ISVD enjoy the luxury of that impression without contest.

Data files

The ISVD had a data file for each of the previously noted four areas of living: occupation, education, military service, and family. Data in each file ranged from general to specific. In addition, data attempted both schematically to represent the present and to outline the future for a decade or so, such outlining being in small time-increments. These specifications obligated the System both to deal with local job markets and to incorporate data on local job vacancies which were helpful in placement suggestions.

The fifth data file in the System contained inquirer characteristics. This file was in two parts. One part dealt with characteristics of inquirers in general and reported relationships of these characteristics with later choices and successes of those inquirers. This file was used both to suggest alternatives to users who needed wider scope for consideration and to subject aspiration to the test of "reality" when the user was in a condition of clarification of a preferred alternative. The other part of the inquirer characteristic data file was the private educational and occupational history of the user as portrayed in his context of developing justification for his preferences and their pursuit and consequences.

Decision-making: The paradigm for choosing

Reflection upon facts/data of the several areas was encouraged with the expectation that the facts/data would be put to personal use, and the user was expected to become guided by the Tiedeman and O'Hara paradigm of vocational decision-making.[14] The paradigm essentially conceives decision in relation (1) to the passage of time, and (2) to the undertaking of the risk and activity required to achieve what one elects to achieve. This conception permits division of the time interval into a period of anticipation and a period of accommodation. Anticipation occurs before the activities of a discontinuity become required; accommodation occurs after activity is required. Stages of exploration, crystallization, choice, and clarification are distinguished within the period of anticipation. Stages of induction, reformation, and integration become possible within the period of accommodation. Distinctions among these stages was made a central part of a MONITOR computer routine in the ISVD.

Computer routines

Computer routines and supporting materials were fashioned to conform with expectation that this vocational decision paradigm both existed and could become explicit and useful to someone who practiced its use. The paradigm determined the computer routines which were developed to permit access to each of the data files and to provide data upon request. There were three primary computer routines: REVIEW, EXPLORATION, and CLARIFICATION.

The REVIEW computer routine permitted call-up and comparison of a prior statement about a then future event both after that expected future event had occurred and after the user had provided indication of how his prior expectations were fulfilled before he sees his prior statement of those expectations. The procedure expected a person to experience insight with regard to consistency and inconsistency available during comparison, and to learn from such insight that his own intuition guides his activity. The intended outcome of REVIEW was that the user learn from his history.

The EXPLORATION computer routine allowed the person to rove through a data file as near randomly as possible. The routine encouraged use of randomness largely at only general levels of materials in order to conserve time but did not forbid specific exploration if, and when, desired. Furthermore, routines were developed to suggest alternatives on the basis of comparison of personal characteristics with established associations between such characteristics of others and their preferred alternatives. The intended outcome from this routine was (1) emergence of a set of alternatives, and (2) the bases on which the alternatives are preferred. I emphasize this latter point in an effort to increase awareness of the reasoning process that is actually involved in career development.

The CLARIFICATION computer routine was available after specific alternatives were selected. CLARIFICATION took the user into queries about the depth of his knowledge concerning then favored alternatives and the understanding of future alternatives which are likely linked with present preferences. The outcome desired was the dispelling both of some doubt and of some ignorance concerning the next step in the progress of career which the person is evolving. Lessening of both doubt and ignorance is likely to increase the user's confidence in meeting the required activities of his next step.

In addition to the three primary computer routines, MONITOR was available as the only secondary computer routine. It essentially consisted of the evaluations which we were able to concoct to determine existence of mastery of stages in the paradigm of vocational decision-making. For this reason, it had to play back into, as well as over, the computer inputs which the person generated. There were three essential aspects of MONITOR. The first was a procedure which we concocted and programmed the computer to provide. The second was the bases on which we caused our judgments to operate among the data put in by the person during his interaction with the computer. The third was the basic computer routines themselves which the person was taught to use if and when he desired to have them. This latter aspect made it possible for the user to write his own monitoring bases to some extent and to have these monitoring procedures play among his material, as ours did originally. I hoped through MONITOR to encourage mastery of the concept of feedback and to give practice and supervision in its application.

Materials

The computer routines incorporated the vocational decision-making paradigm. I did not expect the computer itself to mature fully the capacity and confidence for use of the decision-making paradigm. I therefore designed two other activities into the System in its totality. One of these was the simulation of decision-making. Simulation was available in (1) games, (2) booklets in which the concepts were taught, and (3) decision problems of a vocational nature which had to be solved in interaction with the computer.

The second of our other activities, which I hoped would further mature the use of the paradigm of vocational decision-making, was the provision of responsibility for work under laboratory and practice conditions. In laboratory practice, reality can replace imagination if there is intentful supervision of users as they practice. This supervision was of the same nature as that employed by counselors with users as they are engaged in the simulated activities of vocational decision-making during the user-computer interactions.

Career: The maturation of personal responsibility through vocational development

I have attempted to show that the ISVD expected choice and cultivated the capacity for, and confidence in, choosing by giving users an almost infinite possibility for the exercise of decision-making among data files while simultaneously attempting to make the processes of decision-making both explicit and mastered. These are elements in vocational development which are generally neither unified in this manner nor made available for practice in modes in which complexity is possible but time is not of the essence, at least not the time of persons other than the person engaged in the exercise. The existence of the ISVD therefore created a first-time physical representation of the "outside" which the person must first learn to bring "inside" and then to act toward knowing that it is there but that he need not be "driven" by it if he is the master of it.

In its totality, the ISVD represented "reality" in its data files, offered processes for working with facts/data through its primary computer routines, and provided practice under supervision through (1) its secondary computer routine, (2) its simulation of decision-making, and (3) its personal supervision (a) by a counselor of the person in interaction in the computer routine, and (b) by a vocational educator as the student user assumed real work responsibility in laboratory and practice work situations.

Can a machine counsel?

Allan Ellis immeasurably advanced my understanding of the mediational processes which had to be built into ISVD by seriously addressing with me the question: Can a machine counsel?[15] The reasoning pursued in addressing that question led to my more certain understanding that it is the decisional process itself that a person must come to understand in vocational decision-making, not just his vocational decisions. Since the argument has been published more fully elsewhere, I merely summarize the argument here since I do need the summary for credibility of my later assertions.

Ellis and I cannot even completely specify the procedures necessary to create such a machine. Therefore, either a counselor must counsel as he does now or students must be educated to live as if they counsel themselves, as we preferred and advocated in that paper.

The argument starts with the proposition first that the ultimate goal in counseling is to help another come to ever more thorough understanding of his processes of problem solving. Then think of counseling in relation to so called knowledge *as if* counseling is to *use* knowledge. In doing so, define counseling problems *as design problems*, namely problems associated with the "as if" use or purpose of an operation and its better understood assimilative strategies.

Next assume that the goals of a Counseling Machine are those of counselors, thereby making the goal in both enterprises identical to that extent. If the purpose of counseling is to cultivate man's capacity for personal problem solving as then argued, then the content and the process of counseling must become one for the student just as Bruner[16] contends for education itself. Ergo, counseling is a paradox.

The principle product of counseling is the matured capacity to guide one's self. Counselors themselves *formally* attempt to help a student to counsel himself. Hence, there is a seemingly simple solution to the paradox; the helper or counselor must have *some* initiative *but* the student must maintain control.

Next, analyze the question, "Can a machine counsel?" as means of specifying *how* the counselor might have initiative while the student maintains control. A machine and a human need neither be nor act alike to warrant belief that a machine can ground education in research. All that is required is indication that the machine and the counselor *have the same goals.*

Counseling and its machine have identical goals if, when a person has a problem related to his understanding, he could be equally well sent to either. Understanding is a time extended working out or articulation of self in problem. The mechanism for the working out of self in problem and thus for the inscription of understanding, is the activity of deciding and the problems with which counselors really ought to concern themselves are, therefore, those of deciding. The process of deciding is distinguished by aspects of anticipation and of accommodation with potentially notable steps being in each of the aspects. In enunciating the aspect of accommodation, one thing to which the individual must accommodate in decision-making is the *decision process itself.* In the most general sense, before we would be willing to say that a person has been counseled by machine, this machine would have to accomplish at least three things. First, it would have to reflect the elements of decision-making about self in career problem in such a way that the language of the process was exposed to the student. Second, the machine must encourage the development of awareness of the process of articulating the decisions in the problem and the relation of self to problems as viewed by that process. Finally, the machine must allow and foster the individual's accommodation to the decision process both in terms of specific predicament and, more importantly, in terms of the process in general.

The ascription in the above model of counseling of the capacity for something *in the person* to be both object and subject is what causes the counselor difficulty ordinarily one step removed from the student being counseled and, therefore, an even more insidious difficulty unless closely watched. We must, therefore, take considerable pain to associate counseling with this primary process *in the individual,* not just in the "helper" or counselor.

Can a machine X a Y in Z?

Vistas loomed before me as I began to understand the power of the logic to which Ellis had introduced me. I began to understand that in answering a question such as, Can a machine X? I was not necessarily building a machine, merely engaging in the exercise of specifying the procedure I would use were I to fulfill the goals of X. And I was doing so sufficiently explicitly so that another could know not alone what I was doing, he would know what I was not doing in terms of what he wanted the procedure to do and/or accomplish. But most important of all, I was writing a procedure in which a person interactively came to comprehend X. The question, Can a machine X? therefore became a favored means by which I began to tackle a number of vexing interactive or thought problems.

My first extension of the question, Can a machine counsel? was an attack on the question, Can a machine develop a career?[17] This paper laid out my then current thinking about a theory of career development. The theory was a lot like I have written in describing ISVD above. But the paper also extended the form of the question, Can a machine X? to that of Can a machine X a Y? The form proved up to extension. The issues evolved around what purposes you wanted the machine to fulfill. If you stated those purposes with care, seeing to it that you left comprehension of problem forming to the interacting person himself, the problem still stayed that of comprehending the decisional process itself during times of decision, not just the decision itself.

The next extension of the question, Can a machine counsel? was to the question, Can a machine admit an applicant to continuing education?[18] That paper laid out my current thinking about how the admissions decision might be collaboratively attacked by college applicant and admissions officer, not just unilaterally attacked by the admissions officer himself. This extension stretched the Can a machine X? question into a Can a machine X a Y to Z? question. New theory in measurement becomes possible within such a context. So does new theory in collaborative decision making. And the logic of Can a machine X? remained valid so long as I kept the focus on a decisional process which would itself be comprehended to the mutual satisfaction of two parties, not just applied unilaterally by one.

In the last few years, I have extended the logic to the question, Can a machine ground education in research? I approached this question of the form, Can a machine X a Y in Z? in two phases. In the first phase I wrote "Research and an Education Machine."[19] This paper adapts the logic of "Can a Machine Counsel?" step by step and outlines a so called Research Machine which would become an ISED (Information System for Educational Decisions). I next extended the argument in "An Educating Research Machine Game."[3] This paper essentially presumes that the logic of the Research Machine is generally available—as it should have been had the John Dewey Volume come out in 1971 as expected—and proposes that we continue to program for the imaginary Educating Research Machine as a game if we can't get together enough financial resources to program an Educating Research Machine itself. In both of these papers I proposed that an ISED (a *hypothetical* Information System for Educational Decisions) would need to program the capacity to create IDM's (instructional decision models) within the dual capacities (a) of ERIC (Educational Research Inquiry Centers) in which an IDM could be tacitly implied from its literature, and (b) of interactive logical analyses in which the functions of data construction for IDM testing could take place.

A major problem in education is that of delegation, the freely offered and freely accepted responsibility for knowing. In designing the research-grounding educational machine, we must, therefore, carefully watch the processes by which a person can inform himself while making his educational decisions in interacting with an IDM programmed along with educational research literature and numeric data as well. The ISVD is an example of how this informing process must be further programmed as the higher order programming about the informed self acting upon thought-grounded decisions. In the ISVD example, we can more explicitly see the substance and functions which the research-grounded education machine would have to offer the student if it were to fulfill its assigned purpose of making the student an applied scientist of the art of living. These functions include a substance (machine capacity needed is data files) which could be queried and used to inform one's percepts (machine capacity needed is scripts) so that accommodation of the decision-making process in that substance (machine capacity needed is access routines) would ensue and be used with effortless ease in the system (machine capacity needed is system programming). In relating the ISVD example to the research-grounding education machine, we should thereby partially design the latter so that it has the functions of the former in relation to assimilating the known.

At the present time, I am engaged with John Peatling in writing a monograph which we are tentatively entitling, "A Group Theory of Constructionist Personality Reconstruction." This monograph draws together much that ISVD has birthed. The monograph begins with "Research and an Education Machine." It then goes through a series of chapters in which John Peatling has opportunity to present a mathematical group theory of personality reorganization. In this theory, a group of 16 personality traits are presented which have, through empirical research, proved themselves to satisfy the properties of a mathematical group. Furthermore, Peatling extends the framework of the group to show how new elements can be brought into the group through the self to make it a group of 32, 64, etc. elements. The monograph concludes with the conception of hierarchical restructuring which is applied to the self. The last chapter uses what has now emerged for me as the proven logic of the question, Can a machine X a Y in Z? to address the ultimate personal question, Can a machine ground self in personality?

A proposal—initiate and play a life career machine game

What currently exists as a Life Career Machine, itself is impressive although still relatively trivial. True, there *are* several programs for interactive career guidance and a proposed theory of interactive career development. There are several programs for interactive numerical analysis which go far in helping the educational researcher to solve his problem by relating to his data in more penetrating ways. However, not many people have dared challenge educational research in terms of its capacity to illuminate the problems and understandings *of students* so that *they* grow in self-correcting ways from the improved judgmental functions and structures of others. The achievement of that objective requires the giving of the powers of numeric analysis to students themselves, not just to educational researchers. The writing of computer programs which give students and citizens the capacity to conceive problems as educational researchers and responsible citizens do, when students and citizens are supposed to be in the step of induction in the deciding process, would go far in improving the education of all in an ISLD-like machine, an Information System for Life Decisions. However, the potential would essentially have to embed an existing or revised computer program for computing multivariate and other statistics in the general linguistic decisional framework of an ISED. Hutchinson[20] made a step in this direction when he programmed for the ISVD a procedure which would have given students in the system opportunity to specify conditions which they wanted to fulfill in their choices of occupation and/or education and then reported from variables and observations available in the system the discriminating probabilities of a student's fulfilling his desires.

Although Peatling and I obviously are far from having a self-grounding personality machine, we think that we have proposed a generally interesting possibility for career education, educational research, and self in personality through our analysis. The possibility is to start with a definition of self in personality intended simultaneously to inform the person *being* analyzed, not just the analyzer engaged in the investigation. Furthermore, by acting *as if* a machine could be constructed which would ground self in personality within the outlined purpose, we would essentially force ourselves to specify as best we can and whenever we can what that machine would have to be like. We would thus take the obvious step of acting upon realization that computers are not one but many machines and that the writing of procedures is in effect specifying machines. The machines may not themselves exist and perhaps need never exist. We only ourselves need to think that they *could* exist when they satisfy the functions outlined for them and take the forms programmed for them. Therefore, what we would actually do, which we consider of far more value, is to *discipline ourselves to specify a procedure which might elaborate our understanding of giving meaning to grounding self in personality, thereby figuratively closing the logical gap between the two.* As we do so, we must expose the procedure as well as the understanding. Understanding must be our end thought, procedure our means whereby. Psychology may well be our product, particularly an embryonic psychology of the tacit mechanism of comprehension—including the comprehension of career, counseling, educational research, and self in personality.

The most fundamental premise of our argument is that accommodation of a particular decision is not only itself possible *but that accommodation of the general decision process in a subject and in comprehension is itself possible.* Accommodation of the general decision process itself is a phenomenal feat both literally and figuratively. Accommodation of the decision process itself requires that one is able to comprehend the principles of design in purpose well enough to deal with them somewhat as objects while being beset on all sides during decision by their subjective effects. However, the realization of this power seems to open one to the interactive life which is now so much in demand and even seems so much more frequently present among us these days. Reasoning with thought *as if* there were a machine in man seems to me to bring this potential of man closer to the surface than has heretofore been possible. However, as you determine if I did that or not, set *your* understanding as your end thought goal, use my procedure as your means whereby, and comprehension of epigenesis in decision-making development may well be the synergetic product for yourself—if you interact privately with your doubts sufficiently to understand, of course.

Some of the capabilities of computers are now nearing manlike proportions. Computers, of course, will never be men. However, an understanding of how computers function may well raise for us the question of how much of a machine man really is. Comprehension rests in questions, not answers. Let ourselves question—". . . dividing in order to combine, combining in order to divide—and simultaneously."[21]

THE ROLE OF THE COMPUTER IN INTUITION AND SELF DEVELOPMENT

I undertook this exercise to show you the role of the computer in intuition and self development. I deliberately started my exposition by describing the simple involvement of the computer in an information system we are all familiar with, the ERIC information system. I then went on to note the effects which Walz and Rich predicted that an information system has in general and would specifically have on counselor education. I next proposed that the effects Walz and Rich suggested for educational research in general and for counselor education specifically were too precious to be limited to researchers and counselor educators. I therefore proposed that we help them to be realized by every person by arranging the right adjuvant relationship between persons and facts/data, namely that of mediation. I proposed that the ISVD which I next described in its theoretic detail was an example of such a needed relationship.

I took off from the relatively specific detail of the ISVD into realms of the mind which can be conjured up when you design a system which has the purpose of facilitating the comprehension of the decisional process itself during times of decision. I suggested that you could advance from considering the question, Can a machine X? to the question, Can a Machine X a Y in Z? without invalidating the logic of the paper, "Can a Machine Counsel?" I proposed that my current extension of that question was to comprehension of the machine which is in each of us and can with appropriate programming bring about a grounding of self in personality.

I have obviously left the practical realm far behind me by now. It therefore remains for Katz, Harris, and Blasch to return you to the realm of reality. Katz will do so by treating his SIGI as an example. Harris will do so by treating her CVIS as a second example. Both of these systems have compatibility with the design of the intended ISVD. They therefore remain as operating illustrations of the rudimentary circumstances which ISVD was designed to bring about. Blasch concludes by considering computer-based computer systems of the future. I trust he mentions Discover as a new system in that future potential. Discover is the offspring of CVIS which comes closer than any other system to approximating the extensiveness of ISVD. In doing so it will also come closer than other systems to facilitating the comprehensions of the decisional process while helping another with his decisions. This is the condition needed to cause comprehension of the value of intuition in self development.

REFERENCES

1. Harris, JoAnn (ed.), *Tested Practices: Computer Assisted Guidance Systems*, National Vocational Guidance Association, Washington, D.C., 1972.
2. Tiedeman, D. V., "Self and Hierarchical Restructuring or Harmonizing the Information Process and Self," *Character Potential*, 1973, 6, pp. 149-162.
3. Tiedeman, D. V. and A. L. Miller, "An Educating Research Machine Game," in Handy, R., (ed.), *Education and the Behavioral Sciences*, 1973, in press.
4. Polanyi, Michael, *The Tacit Dimension*, Doubleday, Garden City, N.Y., 1966.
5. Walz, G. R. and J. V. Rich, "The Impact of Information Systems on Counselor Preparation and Practice," *Counselor Education and Supervision*, May 1967.
6. Tiedemann, D. V., "Information Generation: From Data Retrieval to the ISVD," *CAPS Capsule*, Spring 1968, 1, 1- and 10.
7. O'Hara, R. P. and D. V. Tiedeman, "Occupational Facts and their Use: Mediation and the Generation of Occupational Information," in Somers, G. G. and Kenneth J. Little (eds.), *Vocational Education: Today and Tomorrow*, Center for Studies in Vocational and Technical Education, University of Wisconsin, Madison, Wisc., 1971, pp. 63-97.
8. McLuhan, M. and Q. Fiore, *The Medium is the Massage*, Bantam Books, New York, 1967.
9. Polanyi, M., *The Study of Man*, Phoenix Books, University of Chicago Press, Chicago, 1958.
10. Flavell, J. H., *The Developmental Psychology of Jean Piaget*, Van Nostrand, New York, 1963.
11. Neisser, U., "The Multiplicity of Thought," *British Journal of Psychology*, 54:1, pp. 1-14.
12. Bruner, J. S., *Toward a Theory of Instruction*, Harvard University Press, Cambridge, Massachusetts, 1966.
13. Kubie, L. S., *Neurotic Distortion of the Creative Process*, Kansas University Press, Lawrence, 1958.
14. Tiedeman, D. V. and R. P. O'Hara, *Career Development: Choice and Adjustment*, College Entrance Examination Board, New York, 1963.
15. Ellis, A. B. and D. V. Tiedeman, "Can a Machine Counsel?", in Holtzman, W. H. (ed.), *Computer-Assisted Instruction, Testing and Guidance*, Harper and Row, New York, 1970, pp. 345-372.

16. Bruner, J. S., *On Knowing,* Harvard University Press, Cambridge, Massachusetts, 1962.
17. Tiedeman, D. V., "Can a Machine Develop a Career? A Structure for the Epigenesis of Self-Realization in Career Development," in Whiteley, J. M. and Resnikoff, A., *Perspectives on Vocational Development,* American Personnel and Guidance Association, Washington, D.C., 1972, pp. 83-104.
18. Tiedeman, D. V., "Can a Machine Admit an Applicant to Continuing Education?" *in Commission on Tests, Report of the Commission on Tests. II. Briefs,* College Entrance Examination Board, 1970, pp. 161-182.
19. Tiedeman, D. V., "Research and an Education Machine," (to have been in Hedges, W. D. (ed.), *Education and the New Technology,* (the 1970 Yearbook of the John Dewey Society for the Study of Education), a project which was finally aborted in 1973). The author, Northern Illinois University, College of Education, DeKalb, Illinois, 1971.
20. Hutchinson, T. E., *Level of Aspiration and Statistical Models Applicable to the Problem of Refining Choice Bases for Career Development,* unpublished doctoral dissertation, Harvard Graduate School of Education, Cambridge, Massachusetts, 1968.
21. Richards, I. A., *Speculative Instruments,* Harcourt, Brace and World, New York, 1955.

Use of computer in relation to critical guidance factors

by RUSSELL N. CASSEL

University of Wisconsin
Milwaukee, Wisconsin

The single most critical of all guidance factors must of necessity deal directly with manpower national resources, and this always means getting the right man in the right job.[1,2] Indeed, the productivity of a nation's manpower derives alone from such a consideration. Much more important than productivity is the morale of people, but which derives directly from their own personal efficiency in productivity, and which under the principles of humanistic psychology leads to their own self-actualization. Then too, the most cherished possession of men everywhere remains human freedom, and the only real freedom that man ever knows is the degree to which he becomes self actualized, or the degree to which he makes use of his own full potential for productivity.

As a nation progresses from a typical agrarian economy to a characteristic industrial economy, the number and complexity of job career areas necessary increases significantly. As the number and complexity of job career areas increases, the need for more effective career guidance of individuals becomes the more critical. Today The United States Department of Labor *Dictionary of Occupational Titles* lists approximately 65,000 different job areas. No guidance counselor could conceivably remember precise details of even a small fraction of that colossal listing. The computer, on the other hand, is able in the most precise manner possible to retain an unlimited number of job area requirements, with the most current up-dating possible.

Curricular offerings in both the elementary and secondary schools of our nation can be relevant in the minds of students only to the degree to which they perceive such activities as useful in their personal and social living. The single using agencies of the schools remain the employers who hire the student graduates, and the worth of an employee must in the final analysis be measured in terms of the marketable skills possessed. Thus, the expected outcomes of effective vocational guidance must be measured in terms of the number and quality of marketable skills possessed by student graduates.

At the beginning of the 12th grade in the U.S. characteristically more than about 80 percent of the students vow that they will go on to college, but typically fewer than half that number matriculate in educational pursuit beyond high school the following year.[1] More than half of those who actually matriculate in post-high school educational pursuit even a year later have no clear idea of their career goals being prepared for. At the end of the first year after high school one out of every three students insist that their career goals established while in high school were no longer acceptable.[3] These are the kind of facts that the U.S. Commissioner of Education, Sidney P. Marland, Jr. is referring to when he maintains, "We are embarrassed by many ugly statistics."[4]

CRITICAL GUIDANCE FACTORS

Vocational career guidance is not something that exists independent of either the curricular offerings in a school, or of the students involved in the learning process and activity. Rather, it is something that is intimately related to both of these phenomena.

Humanistic psychology

The principles of humanistic psychology establish the context in which effective career guidance alone can emerge; for at the near center of the guidance process there is a very unique human being that makes a difference.[5,6,7] Each of these principles serves to set the stage and establish perimeters, and to violate any one of them serves only to weaken the career guidance process. This is not intended as a discourse on humanistic psychology, but the basic principles are highly germane as critical guidance dynamics:

(1) Feelings Critical—feelings are paramount and derive largely from hidden communications of people.
(2) Focus on Purpose—all human activities are designed to achieve well established goals or needs based on personal satisfactions.
(3) Need Hierarchy—more basic needs in man must be reasonably well met before one is able to deal with cognitive and social areas.
(4) Intrinsic Value of Activity—activity must be an "end" with "intrinsic" value; not a "means" with "extrinsic" value.
(5) Personal Growth—learning must lead to personal growth of the individual and serve to increase personal welfare of individual.

(6) Autonomous—man is a "free will" agent, but competency in decision process alone can insure this status.
(7) Love of Knowledge— permits individual to unfold, become undressed, and scrutable both psychologically and spiritually.
(8) Gaming Spirit—seeks to capture the play spirit of the child with all of its characteristic spontaneity and sincerity of purpose.
(9) Identification with Men—strong and enduring interest in the continued improvement of man's welfare.
(10) Strong Democratic Values—rich and poor, black and white, old and young all are alike, and the group determines the rules for play.
(11) Fresh Appreciation of People and World—people not dulled with feeling that each new experience is same old hat, but stimulating and new.
(12) Self-Actualization—man is truly himself only when he is making full utilization of every potential for own self destiny.
(13) Transcendental Meditation—man is able to utilize new levels of reality and capable of mind control through biofeedback techniques.
(14) Competency Learning—"mastery" deals with "knowledge about" but "competency learning" proceeds to "experience with."
(15) Human Freedom—ranges on scale from "freedom from want" through self-actualization to space for freedom of movement with self-awareness.

Decision making competency

Until and unless an individual has developed decision making competency one cannot be expected to deliberate on career planning; any more than one could be expected to render a concert without the development of musical skills. Failures in vocational guidance must be attributed directly to competency in decision making of the individual; for this is the vehicle alone that generates; not the notions of one's parents or the guidance counselor.

It is not mere coincidence that drug abuse, delinquency, and crime have been increasing in somewhat the same degree as the complexity of jobs have increased. There is every evidence to suggest that such atypical behavior is a symptom of real and progressively increases in failures of career guidance in our schools; that the 70,000 guidance counselors are not without blame in this regards. If we are to deal with accountability in this regards we can be assured that persons lost in the activities of gainful productivity are not likely victims of drug abuse, delinquency, and crime.

Precisely the same decision competency that is a necessary requisite for effective career planning is equally a necessary requisite for the prevention of drug abuse, delinquency, and crime. The continued progressive increase in atypical behavior (drug abuse, delinquency, and crime) attests boldly to the real empirical absence of decision making competency on the part of individuals, and one cannot psychologically expect such behavior to be changed even a mite without improved competency in decision making. The "Back to religion" movement, so characteristic of a Christian nation like the United States, can only serve as a temporary relief; for it promises the individual that someone outside of the individual will be responsible for control. It was Christ who took the single talent from the lady who failed to use it, and gave additional talents to the one with the many talents who used them.

Decision making is learned behavior, not unlike every other psychological process, and it is somewhat akin to similar mathematical processes typically learned in the school. The decision making competencies of individuals may be improved through systematically organized "mastery" and "competency" activities and experiences; not unlike that of every other human competency. It is the most critical of all human learning, for it serves as the fulcrum on which human freedom is attained and achieved. It is the single vehicle by which man must seek to gratify his higher needs and achieve self-actualization; there can be no substitute.

Career guidance function

Individuals in a Democracy have a right to the opportunity to be able to discover and identify their strong and weak talents, and to be able to relate such discoveries to meaningful career planning. Each and every citizen of a Democracy has the right to expect to consider the full repertoire of job careers available in the narrowing of career planning choices. This, then, is the nature of career guidance we have come to know as a professional and scientific discipline.

Where the relevancy of curricular offerings are challenged by large numbers of students it must be considered to represent a failure in career planning and guidance. Whenever the concept of relevancy of curricular offerings is raised, there must be career guidance to resolve the issue. No other logic will suffice. This, to be sure, does not mean that avocational skills and avocational pursuits are unrelated to marketable skills; for anything that pertains to the welfare of the individual contributes directly to his own personal productivity.

If, as Jung maintained, back of every successful man there is a woman, and back of every successful woman there is a man (each the personal image of an ideal mate); so back of every successful and happy family there is a successful job. Not unlike drug abuse, delinquency, and crime the progressive increase in divorce is just another symptom of our failure in career guidance. The problem of relating personal interests, aptitudes, and values to job requirements constitutes the process of career guidance as we know it in the present state of the art. Computer based programs promise to improve this process immeasurably.

Educational guidance function

In an industrial economy where complexity progresses uninhibited the educational guidance follows logically to career planning. Indeed, the one cannot be conceived independent of the other. It is becoming increasingly obvious from the work of Benjamin Bloom and Kenneth B. Clark that educational opportunity in the past has established a myth of "individual differences" in the intelligence of individuals of alarming proportions.[8] If, as Bloom and Clark attest, every healthy normal individual is capable of perfect mastery in learning situations, where teachers know how to teach them, the concept of educational guidance takes on new and unheard of dimensions. More and more we come to realize from emerging research[3] that the social climate, attitude, and context in which behavior occurs is far more important as a causative dynamic of success or failure than what has been traditionally attributed to intellectual capability of individuals.

Human relations

Last, but not least, a critical factor in career guidance is the human relations of people. The great mass of promotions in both industry and government are based largely on how well one gets along with others; not on how competent one is technically. This, to be sure, is as one would expect under the principles of humanistic psychology. The heart of human relations must deal with first understanding what causes people to like each other, and the usual consequence of such behavior. Second, what causes people to dislike each other, and the consequence of such behavior. Indeed, human relations must be considered to be another indispensable marketable skill that is developed; not unlike that of decision competency. We have not placed too much emphasis on academic barriers in the past, but we have failed to place enough emphasis on aspects of personal development, and have too often failed to go beyond mastery learning (knowledge about) to competency learning (experience with) in relation to much of what is important in the life space of individuals.

DEDEV

The Computerized Decision Development System (DEDEV) is designed as a means for developing decision competency in individuals. It is comprised of 14 regular modules and six supplementary modules for use as a credit course in high schools or college. Each separate module is an independently organized program of experiences dealing with different dimensions involved in the decision making process.

Module I—Introduction to DEDEV.
Module II—Humanistic Psychology.
Module III—Locus of Control.
Module IV—Functions of the Ego.
Module V—Models of Excellence.
Module VI—Systems Analysis.
Module VII—Vector and Valence Analysis.
Module VIII—Conscience and Ego-Ideal Development.
Module IX—Deliberation Literacy Hierarchy.
Module X—Group Leader Decision Pattern.
Module XI—Power and Decision Making.
Module XII—Organizational Climate and Management.
Module XIII—Decision Counseling in Helping Relationships.
Module XIV—Dynamics of Human Freedom.
Module XV—Levels of Human Freedom.
Module XVI—Functions of Human Brain.
Module XVII—Steps in Transcendental Meditation and Mind Control.

"O-P-A-H-U-D-E" method utilized

The method utilized for implementation of DEDEV makes use of a technique referred to by the acronym "O-P-A-H-U-D-E," with each letter in the acronym representing a separate aspect of the procedure. The method makes use of both a conventional and an nonconventional approach.

Conventional approach—This aspect makes use of the first four letters in "O-P-A-H-U-D-E," and with the letters standing for the following activities:

O—Orientation—where individuals are informed as to expected outcomes, and where the stage is set for such outcomes to emerge.
P—Presentation—where a half hour video tape by an authoritative person presents the theoretical discussion of MODULE.
A—Assimilation—where teacher or paraprofessional discusses theory presented for relating it to experience of subjects.
H—Humanization—where theory is related to persons through the experiences of the participants.

Nonconventional approach—This aspect makes use of the last three letters of "O-P-A-H-U-D-E" and with the letters standing for the following experiences and activities:

U—Utilization—where computer-based gaming and simulation assigns individuals to surrogate roles at cutting edge of confrontation with "meaty" social problems.
D—Differentiation—where participants are assigned similar surrogate roles in precisely same social meaty problems, but this time as confronters or exploiters.
E—Evaluation—where the computer provides immediate knowledge of progress or success with knowledge of degree of such success.

Mastery to competency

Typically, the conventional approach utilized in O-P-A-H-U-D-E is expected to yield information about, or mastery of theory in relation to the various dimensions or MODULES of DEDEV. The nonconventional approach seeks to provide empirical experience in the utilization of such newly presented theory in a variety of contexts, and ranging from use of confrontation to being confronted. A "meaty" social problem for purposes of DEDEV is defined as an exaggeration of the typical conditions, and representing situations that can and do happen, but that would be expected to be the exception to the rule. Range of choices provided always include positions of (1) conformity, (2) antithesis of conformity, and (3) middle-of-road. In addition, there are two other choices representing varying degrees of referenced continuum. For every choice both hazards and consequences are programmed into the computer. Always, hazards represent future risks that are external to individual; while likely consequences represent past tense with impact present on individual. Correct answers programmed into computer range from normative base for corresponding persons (typical persons 12 to 18 years of age), to theoretically assigned correct responses based on theory involved, i.e., BASPAT assigned each of four answers as being (1) autocratic submissive, (2) democratic parliamentarian, (3) autocratic aggressive, or (4) laissez faire, and depending on the definition given by Lewin in his now famous Iowa studies.[9]

Evaluation of DEDEV

A number of different evaluative studies have been accomplished on DEDEV, some of which are still in progress.[10,11] The first important evaluation was of an informal type accomplished by American Institutes for Research.[12] This study was very favorable and suggested that DEDEV become the nucleus of a doctoral program in the helping relationships area. The second study was accomplished by The Far West Research Laboratory and was equally positive in findings, and it was of the same observational type as was the AIR study.[13] Next came the Kim Wyman informal assessment where he asked to use these programs for a group of colleges in Australia, and where he suggested that DEDEV was one of two such programs in the United States that were ready for school in his written report. DEDEV was a portion of treatment provided approximately 250 parolees in 1968-69 with recitivism being reduced from high in the 60's percentages to low in the 20's percentages.[14] A formal study was accomplished making use of ROTC students at The University of Wisconsin-Milwaukee, and it was equally positive in findings for support of DEDEV.[15] Two separate doctoral dissertations now under way—one with high school students and the other with college students—deal squarely with formal aspects of an evaluation of DEDEV.[27,11]

Computer soft-ware

The computer program for driving DEDEV has been written in a number of different higher level computer languages: (1) FORTRAN IV, (2) FORTRAN V, (3) BASIC, (4) ALGOL, and (5) ASSEMBLER for the PDP-8.

VOCGUYD

The Milwaukee Computerized Vocational Guidance System (VOCGUYD) was developed jointly by Professor Cassel of The University of Wisconsin-Milwaukee and persons in the Guidance Department of The Milwaukee Public Schools (Terry Mehail, Alfred Thurner, and Ralph Onerheim).

Career choice field

All 293 job career areas listed in the U.S. Department of Labor *Occupational Outlook Handbook, 1970-71 Edition* with their primary "shredouts" (instead of one nurse, several nurses of different specialties; instead of one college professor, several college professors of different specialties; instead of one physician, several physicians of different specialties; etc.) were included. This represents a total of 1,187 job career areas, covering the entire range of the DOT (*Dictionary of Occupational Titles*) 22 Job Skill Requirement Areas. This included, for example, 47 jobs in art, 30 in business relations, 85 in clerical, 41 in counseling and social work, 92 in crafts, 37 in education and training, 47 in elemental work, 59 in engineering, 36 in entertainment, 43 in farming and fishing, 189 in machine work, etc.

Search and screening criteria

Seventeen different "search and screening" criteria are included in VOCGUYD in four different areas of concern: (1) Personal Interest of Participant, (2) School Success, (3) Special Aptitude, and (4) Work Values.

Personal interest—Here six different criteria are available, each one dealing with a different aspect of personal interest of the individual, but with a great deal of overlap among some of them: (1) *Kuder Occupational Interest Inventory* areas, (2) Social levels of Goodenough Socio-economic Hierarchy, (3) Data, persons, or things from DOT; (4) school level desired to attain for entry into job, (5) DOT Job Skill Requirement Areas, and (6) Ohio Interest Survey scores. A participant may request to search and screen from among the 1,087 jobs by use of either of these six criteria. If an individual, for example, selects the Kuder, the computer promptly displays the 10 different areas of the Kuder (outdoor, mechanical, computational, scientific, persuasive, artistic, literary, musical, social service, and clerical), and asks which of them is of interest for the narrowing of career choices. The same general procedure would be followed if one of the other six criteria were selected.

School success—This criterion makes use of the individual's Grade Point Average (GPA), and is merely used to determine if an individual's past school success warrants inclusion of semi-professional and professional job areas.

Special aptitudes—This was designed to make use of multiple aptitude test scores for the narrowing of career choices, and includes six of the areas from The Differential Aptitude Test Battery: (1) Verbal Ability, (2) Numerical

Ability, (3) Abstract Reasoning, (4) Clerical Speed and Accuracy, (5) Mechanical Reasoning, and (6) Space Relations. These may be used as the basis for search and screening, or as a means to check one's special aptitude against an interest area selected. If one of the six personal interest criteria are used for search and screening, critical special aptitudes are depicted for jobs selected (meaning that a T-score of 50 or better is generally suggested for success in that area of aptitude).

Work values—The four factor areas of the Super *Work Values Inventory* are used for this area: (1) Material, (2) Goodness of Life, (3) Self-Expression, and (4) Behavior Control. These four areas may be used precisely as described for the special aptitude areas.

Interrogation function

Generally, five questions critical to each of the separate 1,187 job career areas are programmed into the computer. These questions have been compiled by approximately 100 Career Advisory Board Members to the Milwaukee Public Schools; with persons from specific job skill requirement areas writing the questions for their own areas of competence. Each of the questions is weighted in terms of importance to success in the job area. The purpose of questions is to make use of the computer to relate the participant's personal interest, school success, aptitude, or work values to the specific job requirements. After a computer interrogation for a particular job, a Career Success Index (CSI) is provided depicting degree of agreement between the interests expressed by the participant and the job requirements.

Career success index

Individuals are expected to proceed with the VODGUYD search and screening until a minimum of from three to five CSI's of average or better are obtained. Such job areas are then the subject of a more intense and concentrated investigation with the assistance of a Vocational Guidance Counselor. VOCGUYD is not intended to replace the counselor, but to supplement such services.

Fostering career maturity

Where individuals are not concerned with identifying specific career job areas, but rather with the development of "Career Maturity," as is typically the case for the junior high school student, or with elementary school age students, from five to eight individuals may work with a single teletype or Cathode Ray Tube (CRT) as an "I-O" Station to the computer.

Auxiliary programs

A whole series of career related Computer Assisted Instruction (CAI) programs are included with VOCGUYD. For example, each of the 17 search and screening criteria are the subject of such an auxiliary program. Individuals desiring information on such criteria may ask computer for such a CAI program. Similarly, tests typically associated with career planning are also the subject of such CAI units, i.e., GATB, NATB, DAT, etc. Other subjects, for example, include: women, economics, job satisfaction, etc.

EDGUYD

The Milwaukee Computerized Educational Guidance System (EDGUYD) was developed jointly by Terry Mehail of the Milwaukee Public Schools, and Professor Russell Cassel of The University of Wisconsin-Milwaukee. VOCGUYD and EDGUYD are considered to be intimately related programs. Indeed, EDGUYD functions much the same as was described for VOCGUYD. Instead of job career areas for EDGUYD, four year post-high school educational facilities are the subject matter and concern. For Wisconsin, not only four year colleges, but all post-high school facilities of any type are included. EDGUYD includes 1,811 post-high school facilities, with 130 being from Wisconsin. States like Alaska, Wyoming, Guam, and Virgin Islands only have one or two schools; while states like California and New York each have more than 100 (103 and 151, respectively).

Search and screening criteria

There are 12 different search and screening criteria, each one based on a three digit number: (1) State or Location, (2) Type of School, (3) Type of Student Body, (4) Institutional Control, (5) Admissions Policy, (6) Enrollment, (7) Cost, (8) Type of Community, (9) Special Considerations, (10) Type of Degrees Conferred, (11) School Term, and (12) Proximity to Milwaukee.

Interrogation function

Five or more questions have been programmed into the computer for each of the 1,811 schools included. Where an individual selects a school of interest, the computer begins the interrogation along the same lines as for VOCGUYD. Here the objective is to narrow choices from the total available schools to a few for more intense study with the aid of a guidance counselor. The interrogation by the computer is designed to relate the interests of the individual with specific school requirements. Again, each of the interrogation questions is assigned a percentage value or index, with all questions for a particular school always adding up to 100 percent. At the end of each such interrogation, the computer provides an Educational Success Index (ESI) that depicts degree of agreement between personal interest of subject in relation to school requirements.

Educational success index

As with the case of VOCGUYD participants are expected to proceed with the EDGUYD program until they have produced from three to five CSI's of "Average" or better as a basis for more intense and concentrated research and study, and for purpose of working with the Guidance Counselor. Again, the EDGUYD program is not intended as a substitute for the counselor, but rather a supplement.

Auxiliary programs

A number of typical CAI programs related to educational guidance have been programmed into the computer, and may be used in connection with EDGUYD. These CAI programs, for example, include tests dealing with education such as: (1) DAT, (2) Iowa Tests of Educational Development, (3) Meaning of I.Q., (4) Preliminary Aptitude Test Battery, (5) Scholastic Aptitude Test Battery, etc. It includes the "Mastery Learning" concept of Bloom, the "Critical Development Levels" of Havighurst, and a series of programs dealing with securing finances for a post-high school educational pursuit.

Fostering career maturity

Just as with VOCGUYD from five to eight persons can work at a single console with EDGUYD for purposes of developing career maturity. This would be the characteristic manner for use of EDGUYD by junior high school and elementary age students.

HUMRELAT

The Computerized Human Relations Program (HUMRELAT) was designed as a means for developing human relations positive skills in individuals. It is planned as a three credit course for a lower division college program, and may be profitably used by high school students for credit, as well. Eight different MODULES are included in HUMRELAT, each designed for a two week period of intensive study and experiences:

Module I—Social Climate.
Module II—Scientific Decision Process.
Module III—Interpersonal Attraction.
Module IV—Interpersonal Rejection.
Module V—Confrontation and Crisis.
Module VI—Group Process and Balancing.
Module VII—Disadvantaged.
Module VIII—World Social Problems.

Based on research findings

HUMRELAT represents the "end product" of a careful review and study of course descriptions of the 17 different technical institutes in Wisconsin. The teachers of human relation courses in each of the 17 technical colleges were asked to indicate their specific preference for inclusion in such a course from a questionnaire.[16] The questionnaire included 21 major topic headings with 84 subtopical ones. It was compiled from the existing courses in human relations presently features at the 17 colleges.

The eight different modules contained in HUMRELAT are those clusters of related subtopics from the questionnaire receiving the highest ratings by the faculty at the 17 technical colleges. The sequencing of the modules was based on psychological continuity.

"O-P-A-H-U-D-E" method utilized

The "O-P-A-H-U-D-E" described under DEDEV is used with HUMRELAT. Both the conventional and nonconventional aspects are included. Half hour video tapes are available for presentation of each of the modules by an authoritative figure. The development of positively orientated human relation skills is deemed as the most important expected outcome, and which are considered to be indispensable marketable skills in relation to manpower resources of a nation.

CASTY

The UWM Computerized Case Study Analysis (CASTY) is a program designed for use in the "School Psychology Practicum," but may be appropriately used for a wide range of other purposes. Here real live cases are programmed into the computer, and participants may engage in dialogue with such persons as in real life. Participants inform computer from what vantage questions are directed and all previously derived information from that respective vantage will be recalled on the cue of key words or phrases. When questions are directed at the subject, answers are in first person, viz., What is your name? My name is Jennienoread, or Jodifferent, etc. Questions directed at the psychologist, physician, teacher, etc., for example, are answered in third person, viz., Do you know the subject involved in this study (to family physician)? Yes, I have been the family physician, was present at her birth, and know Jennienoread well. CASTY could be used for empirical experience in diagnosing confrontation, crisis, and exploit in every conceivable arena of life.

REFERENCES

1. Marland, S. P., Jr., "Career education: every student headed for a goal," *American Vocational Journal*, 1972a, XLVII, pp. 34-38.
2. Cassel, R. N. and T. Mehail, "The Milwaukee Computerized Vocational Guidance System (VOCGUID)," *Vocational Guidance Quarterly*, 1973a, 21, 3, pp. 206-213.
3. Flanagan, J. C., J. T. Dailey, M. F. Shaycoft, W. A. Gorham, D. B. Orr, and I. Goldberg, *Design for a Study of American Youth*, New York, Houghton Mifflin Company, 1962.

4. Marland, S. P., Jr., "Educational communications—the future is now," *Education*, 1972b, 93, 1, pp. 3-9.
5. Buhler, C., "Humanistic psychology as an educational program," *American Psychologist*, 1969, 24, 8, pp. 736-742.
6. Rogers, C. R., "Some new challenges," *American Psychologist*, 1973, 28, 5, pp. 379-387.
7. Cassel, R. N., "Fundamentals of humanistic psychology," *World Journal of Psychosynthesis*, 1973a, 5, 4, pp. 21-24.
8. Bloom, B. S., "Recent developments in mastery learning," *Educational Psychology*, 1973, 10, 2, pp. 53-57.
9. Barker, R. G., J. S. Kounin and H. F. Wright, *Child Behavior and Development*, New York, McGraw-Hill Book Company, 1943.
10. Nault, J., *Computer-Based Decisionmaking Learning to Foster Personal Development in College Students*, (Doctoral dissertation in progress), 1974.
11. Reise, M., *Use of Decisionmaking Learning to Foster Development in High School Students*, (Doctoral dissertation in progress), 1974.
12. Tiedeman, D., *An Informal Assessment of The University of Wisconsin-Milwaukee Computerized Guidance Programs*, Institute for Research in Education, American Institutes for Research, Palo Alto, California, 1972.
13. Keyes, D., *Drug Education*, U.S. Department of Health, Education, and Welfare, National Institute of Education, PREP Report No. 36, 1972.
14. Mehail, T., *Evaluation of The UWM Computerized Guidance Programs with secondary school students*, unpublished informal report, Milwaukee Public Schools, Milwaukee, Wisconsin, 1969-71.
15. Cassel, R. N., and S. D. Stroman, "Evaluation of The Computerized Decision Development System (DEDEV) for use with ROTC students," *Journal of Instructional Psychology*, 1973, 1(1) pp. 12-22.
16. Cassel, R. N., J. G. Atwood and A. C. Lie, "The Computerized Human Relations Program (HUMRELAT)," *College Student Journal Monograph*, 1973, 7, 2, No. 2.
17. Brubaker, D. L., and R. H. Nelson, Jr., *Introduction to Educational Decision Making*, Dubuque, Iowa, Kendall/Hunt Publishing Company, 1973.
18. Cassel, R. N., "Computer orientation," *Newsletter of the Association for Development of Instructional Systems* (ADIS), September, 1971, pp. 11-16.
19. Cassel, R. N., "Fundamental principles for teaching by objectives," *Peabody Journal of Education*, 1972a, 50, 1, pp. 75-80.
20. Cassel, R. N., "The UWM Computerized Decision Development Competency System (DEDEV)," *Psychology*, 1972b, 9, 3, pp. 40-46.
21. Cassel, R. N., "Different types of computer-based interaction modes," *College Student Journal*, 1972c, 6, 3, pp. 74-76.
22. Cassel, R. N., "Helping relationships and decision making competency," *World Journal of Psychosynthesis*, 1972d, 4, 7, pp. 26-30.
23. Cassel, R. N., *The Psychology of Decision Making*, North Qunicy, Massachusetts, The Christopher Publishing House, 1973b.
24. Cassel, R. N., *The Computerized Decision Development System (DEDE V)*, Jacksonville, Illinois, Psychologists and Educators, Inc., 1973c.
25. Cassel, R. N., "The UWM Computerized Case Study Analysis," *Psychology*, 1973d, 10, 2, pp. 6-14.
26. Cassel, R. N. "Psychological aspects of human freedom. *Psychology*," (Pending), 1973e.
27. Cassel, R. N., and L. P. Blum, "The School Dropout," *The Pointer*, 1970, 13 (4), 53-56.
28. Cassel, R. N. and T. Mehail, "*The Milwaukee Computerized Vocational Guidance System* (VOCGUYD)," Jacksonville, Illinois, Psychologists and Educators, Inc., 1973b.
29. Cassel, R. N. and T. Mehail, "The Milwaukee Computerized Edutional Guidance System (EDGUYD)," *Education*, 1973c, 94, 1, pp. 38-43.
30. Grayson, L. P., "Challenge to educational technology," *Science*, 1972, CLXXV, pp. 1216-22.
31. U.S. Department of Labor, *Occupational Outlook Handbook, 1970-71 Edition*, Washington, D.C., 1970.
32. U.S. Department of Labor, *Dictionary of Occupational Titles*. Washington, D.C., 1968.
33. Wyman, K. T., "Computers and educators: some observations," *Keynote Discussion at The Seventh Annual Conference of Association for Educational Data Systems*, Ellenville, New York, October 1-4, 1972.

Effective demonstration of minicomputer-based systems by a novel digital simulation

by SADASHIVA S. GODBOLE

Babcock & Wilcox Company
Lynchburg, Virginia

INTRODUCTION

Minicomputer-based systems are becoming more and more popular. A typical system of this type consists of an industrial process to be controlled/monitored and a minicomputer connected to it through an interface. (See Figure 1.) Effective demonstration of such a system is often required in many instances. For example, it is required in industry during the initial phase of justification (to the management) for the manufacture of such a system. It is also required in education while teaching minicomputer-based gadgets like optimum controller, Luenberger observer, and Kalman filter. Normally this is done through the usual digital or hybrid simulation. However, the usual digital simulation, while capable of providing all the desired information, is not very appealing because it does not physically bring out the conversation type interaction between the process and the computer. This is not true in the case of a hybrid simulation of the system where the process is simulated on the analog part and the minicomputer is played by the digital part. The hybrid simulation, however, requires the availability of a hybrid computer, a fairly expensive equipment, and may or may not be easy to implement depending on the features of the particular hybrid computer facility and the availability of personnel conversant with hybrid programming. Of course, a prototype system using the actual process and the proposed minicomputer is the most direct demonstration of the system but is generally a quite costly approach. Thus there seems to be a lack of an all-digital simulation that has the same appeal as the hybrid simulation and which is interactive and conversational. Such a simulation is developed next.

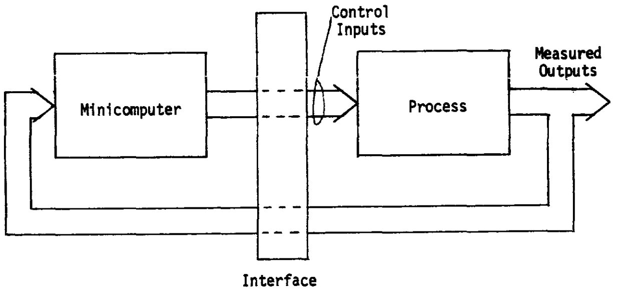


Figure 1—A typical minicomputer-based system

THE NEW APPROACH

Due to its interactive and conversational natures, the new simulation to be developed must necessarily use teletype terminals connected to a, usually remote, time-shared computer facility. Furthermore, in order to have the same appeal as the hybrid simulation, the new approach should use two teletypes, one for the process and one for the minicomputer, each capable of accepting the necessary parameters

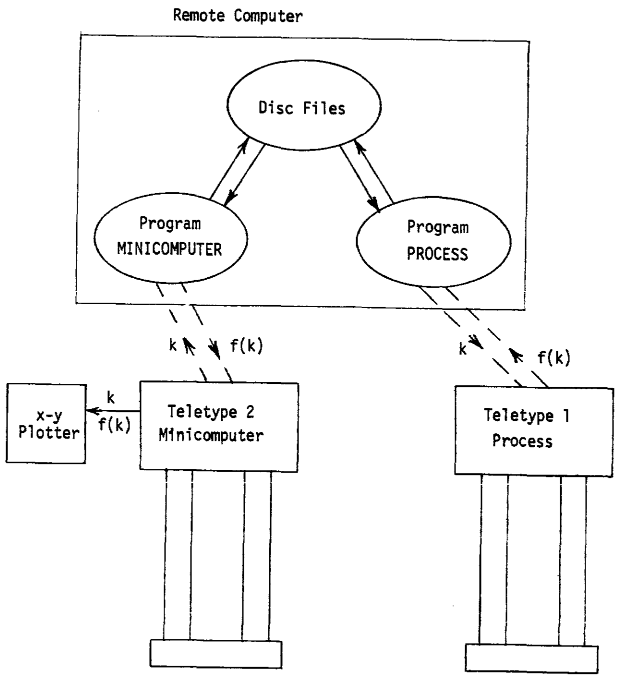


Figure 2—Set-up for the example simulation

```
C PROGRAM MINICOMPUTER.F4
C
C    SUBROUTINES REQUIRED- NONE
C
C    FILES MANIPULATED-
C          1. BINARY FILE KLAST.DAT
C          2. BINARY FILE VALUE.DAT
C          3. SYMBOLIC FILE FLAG1.DAT
           4. SYMBOLIC FILE FLAG2.DAT
C
C    REMARK
C    FILES 3 AND 4 CONTAIN 1 RECORD EACH AND SHOULD
C     BE INITIALIZED TO -1 IN FORMAT(1X,I5) BEFORE EXECUTING
C     THIS PROGRAM.
C
       TYPE 100
     PAUSE 'PAUSE TO SET PLOTTER. PLTL'
C
C    INITIALIZE THE FILE KLAST.DAT
       K=-1
       TYPE 110
       ACCEPT 130, LAST
       OPEN (1,'KLAST.DAT',RANDIO(1),BINARY)
       WRITE (1#1) K
       WRITE (1#2) LAST
       CLOSE (1)
C
  10   CONTINUE
       IFLG1=1
C
C    READ K, F(K) AND PLOT.
       OPEN (1,'KLAST.DAT', RANDIO(1),BINARY)
       READ (1#1) K
       CLOSE (1)
       IF(K.EQ.-1) GO TO 1000
       J=K+1
       OPEN (2,'VALUE.DAT', RANDIO(1),BINARY)
       READ (2#J) FUNCT
       CLOSE (2)
       IX=K*100
       IY=IFIX( 4999.5*(FUNCT+1.0)+.5 )
       TYPE 120, IX,IY
  1000 CONTINUE
       OPEN(3,'FLAG1.DAT',OUTPUT,SYMBOLIC,ERR=1000)
       WRITE(3,150) IFLG1
       CLOSE(3)
       IF(K.EQ.LAST) GO TO 50
  1005 CONTINUE
       OPEN(4,'FLAG2.DAT',INPUT,SYMBOLIC,ERR=1005)
       READ(4,150) IFLG2
       CLOSE(4)
       IF(IFLG2.LT.0) GO TO 1005
       IFLG2=-1
       OPEN(4,'FLAG2.DAT',OUTPUT,SYMBOLIC)
       WRITE(4,150) IFLG2
       CLOSE(4)
       GO TO 10
  50   TYPE 140
       STOP
C
  100          FORMAT(///
      1  ' I AM THE MINICOMPUTER. I READ THE VALUE OF THE '
      2 /' PROCESS OUTPUT OVER A GIVEN INPUT RANGE SUPPLIED'
      3 /' BY ME AND PLOT IT.'//)
  110          FORMAT(1H0,'UPPER LIMIT OF ARGUMENT RANGE=?:',$)
  120  FORMAT (5X,I5,5X,I5)
  130  FORMAT(5I)
  140          FORMAT(1H ,'*** END OF SIMULATION *** PLTT'///)
  150  FORMAT(1X,I5)
       END
```

```
C    PROGRAM PROCESS.F4
C
C    SUBROUTINES CALLED:  NONE
C
C    FILES MANIPULATED:
C         1. BINARY FILE KLAST.DAT
C         2. BINARY FILE VALUE.DAT
C         3. SYMBOLIC FILE FLAG1.DAT
C         4. SYMBOLIC FILE FLAG2.DAT
C
C    REMARK
C    FILES 3 AND 4 CONTAIN ONE RECORD EACH AND SHOULD BE
C    INITIALIZED TO -1 IN FORMAT(1X,I5) BEFORE EXECUTING
C    THIS PROGRAM.
C
       TYPE 100
       TYPE 110
       ACCEPT 200,A
       TYPE 120
       ACCEPT 200,DT
       TYPE 130
       ACCEPT 200,W
       ADT=A*DT
       WDT=W*DT
C
       TYPE 140
C
 10    CONTINUE
       OPEN(3,'FLAG1.DAT',INPUT,SYMBOLIC,ERR=10)
       READ(3,210) IFLG1
       CLOSE(3)
       IF(IFLG1.LT.0) GO TO 10
       IFLG1=-1
       OPEN(3,'FLAG1.DAT',OUTPUT,SYMBOLIC)
       WRITE(3,210) IFLG1
       CLOSE(3)
C
C    OPEN FILE KLAST.DAT
       OPEN (1,'KLAST.DAT', RANDIO(1),BINARY)
       READ (1#1) K
       IF(K.EQ.-1) READ(1#2) LAST
       IF(K.EQ.LAST) STOP
       K=K+1
       WRITE (1#1) K
       CLOSE (1)
       FK=FLOAT(K)
       VAL = EXP(-ADT*FK) * SIN(WDT*FK)
       TYPE 150, K,VAL
C
C      OPEN FILE VALUE.DAT TO WRITE F(K)
       OPEN (2,'VALUE.DAT', RANDIO(1),BINARY)
       LOC=K+1
       WRITE (2#LOC) VAL
       CLOSE (2)
C
       IFLG2=1
 20    CONTINUE
       OPEN(4,'FLAG2.DAT',OUTPUT,SYMBOLIC,ERR=20)
       WRITE(4,210) IFLG2
       CLOSE(4)
       GO TO 10
C
 100          FORMAT(///
      1  ' I AM THE PROCESS. I UPDATE THE PREVIOUS VALUE K-1'
      2 /' OF THE TIME INDEX SUPPLIED BY THE MINICOMPUTER AND '
      3 /' COMPUTE THE VALUE OF THE FUNCTION'
      4 /10X,   'F(K)=EXP(-A*K*DT)*SIN(W*K*DT).'
      5 /' I THEN PRINT K, F(K) AND SEND THESE VALUES TO THE '
      6 /' MINICOMPUTER.'//)
 110   FORMAT (1H ,'SPECIFY A, DT, W IN FREE FORMAT'/1H ,'A=? ',$)
 120   FORMAT (1H ,'DT=? ',$)
 130   FORMAT (1H ,' W=? ',$)
 140          FORMAT(1H0,'TIME INDEX K    F(K)'/' ------------     -----')
 150   FORMAT (1H ,3X,I3,9X,F6.3)
 200   FORMAT (5E)
 210   FORMAT(1X,I5)
       END
```

Figure 3—Listing of programs PROCESS and MINICOMPUTER

and of displaying significant results. The two terminals should also preferably operate as independent units (like two main programs, not as one main program and one subroutine) linked only through the interface of disc files or appropriate device assignments. The above requirements make the new approach non-trivial. Such an approach, satisfying all the above requirements, has indeed been realized and is exemplified by the following simple example.

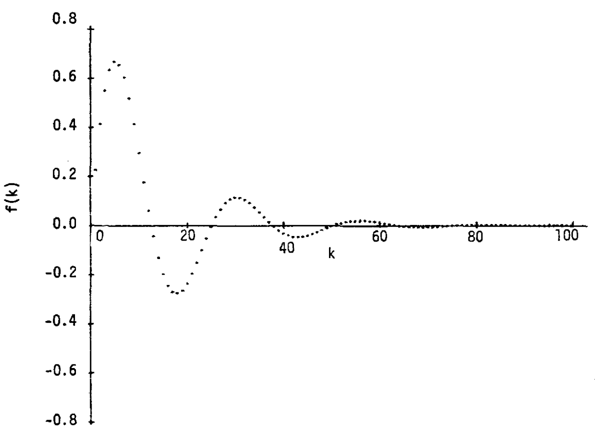
EXAMPLE

Suppose the process is a device which can calculate and print the value $f(k)$ of an exponentially decaying sine wave at any desired time instant k, namely,

$$f(k) = e^{-ak\Delta t} \sin(\omega k \, \Delta t)$$

where a, ω and Δt are the process parameters. Further suppose the minicomputer connected to this process *dynamically* supplies the desired time instant k to the process and plots the function $f(k)$ returned by the process for $k \epsilon [0, N]$. Separate main programs, PROCESS and MINICOMPUTER, were written for use on the Tymshare Inc., California facility to describe the respective functions. The communication between the two programs was established through disc files. These programs were then executed from two teletype terminals, the terminal playing the minicom-

```
RUN MINICOMPUTER


I AM THE MINICOMPUTER. I READ THE VALUE OF THE
PROCESS OUTPUT OVER A GIVEN INPUT RANGE SUPPLIED
BY ME AND PLOT IT.


PAUSE TO SET PLOTTER. PLTL
TYPE G TO CONTINUE,X TO EXIT.
*G


UPPER LIMIT OF ARGUMENT RANGE=?: 100
```



```
*** END OF SIMULATION *** PLTI
```

```
-RUN PROCESS


I AM THE PROCESS. I UPDATE THE PREVIOUS VALUE K-1
OF THE TIME INDEX SUPPLIED BY THE MINICOMPUTER AND
COMPUTE THE VALUE OF THE FUNCTION
          F(K)=EXP(-A*K*DT)*SIN(W*K*DT).
I THEN PRINT K, F(K) AND SEND THESE VALUES TO THE
MINICOMPUTER.


SPECIFY A, DT, W IN FREE FORMAT
A=?   0.7

DT=?   0.1

 W=?   2.5

TIME INDEX K      F(K)
------------      -----

      0           0.000
      1           0.231
      2           0.417
      3           0.553
      4           0.636
      5           0.669
      6           0.655
      7           0.603
      8           0.519
      9           0.414
     10           0.297
     11           0.177
      •             •
      •             •
      •             •
      •             •


EXIT
```

Figure 4—Terminal activity during the example simulation

puter being connected to a Hewlett-Packard HP-7200A x-y plotter. The set up of this simulation is shown in Figure 2, the programs PROCESS and MINICOMPUTER, in Figure 3 and the terminal activities in Figure 4.

CONCLUSION

In conclusion, it can be said that we have developed a new approach to simulate minicomputer-based systems. The simulation is digital, interactive and conversational. Only an elementary digital programming experience and moderate equipment are needed for such a simulation. This approach has potential applications in education as well as industry. Further, the interactive and conversational nature of the new simulation lends itself very well to games played through the computer. The players at distant locations can enter their moves (strategies) and receive the opponents' responses through their terminals.

Twenty commandments for managing the development of tactical computer programs

by JAMES A. WARD

Naval Ordnance Systems Command
Arlington, Virginia

INTRODUCTION

The million-word AEGIS computer program has been successfully completed.

We think that the successful development of this computer program has been due to a number of management decisions and policies used during the life of the contract. The purpose of this paper is to express these principles so that we can be sure to adhere to them during the next phase of AEGIS development and so that others may be able to adopt those applicable in their projects. Obviously many of these management ideas are used by others, but possibly not nearly to the degree as in AEGIS.

BACKGROUND

AEGIS is a shipborne anti-air warfare system consisting principally of a phased array radar (AN/SPY-1), a weapon direction system (MK 12), a missile launcher (MK 26), Fire Control System (MK 99), the SM-2 missile, Control Command System (MK 130), and other related components including a number of AN/UYK-7 computers. Under phase 1 of the present engineering development contract we are building a minimal system to test AEGIS feasibility. This was put together first at a Land Based Test Site (LBTS) in New Jersey and checked out against live targets. It is now being installed in USS NORTON SOUND for tests at sea.

The digital computer program generated for AEGIS is a large and fundamental part of the system. It has always been anticipated that generation and integration of these programs would be one of the high risk technical endeavors in the development of the system. While there have been problems there has been none of any great significance.

A very considerable computer programming effort has been made for EDM-1, the first engineering development model, for it requires four AN/UYK-7 computers, each about the operational equivalent of an IBM 360/65. There are well over a million words of code, half operational and half support.

As this is written at the end of the test at the Land Based Test Site, the AN/UYK-7 computers (there are now 11 under AEGIS control) and their programs are performing very well. However, there have been considerable difficulties in the past. AEGIS was the first project to receive AN/UYK-7 computers. Serials 3, 4 and 5 were delivered in the spring of 1970. Some 6 months later a major change (addendum 4) was made which changed the character of the computer, programming-wise. Then for the next 2 years we had all the infantile problems of a new computer design. The usual manufacturing errors and design deficiencies were found the hard way by the computer programers. The maintenance men also had to learn.

There were more than usual initial problems with computer programming. Essentially the computers came bare.

There were delivered the Ultra 32 assembler, a factory acceptance test, and a few utility programs (load, dump, decimal to octal, octal to decimal). There was no compiler, no operating system, no executive, no standard set of utility programs, and not even a maintenance/fault isolation program. All of these had to be developed. However, the compiler, not developed by AEGIS, gave the most trouble.

The compiler (now know as CMS-2Y) was scheduled to be delivered to the contractor as Government Furnished Equipment in June 1971. We needed it a year earlier, so we accepted as an interim compiler, the CMS-2Q, which operates on the 642B computer to generate code for the AN/UYK-7. This caused some inefficiencies, but in order to meet our production milestones we have continued with it. However we plan to change to the CMS-2Y for the next phase of AEGIS program development.

MANAGEMENT DECISIONS AND POLICIES

Let us now look at some of the management decisions and policies that have made the computer programming successful in spite of these difficulties. No one of these is the reason for our success, however we feel it would be risky to drop any, or even let any of them deteriorate. Certain of them are sine qua non. These will be starred in what follows. We feel that continuation of all of these will assure success in developing the computer programs for the DG, the first AEGIS ship of the fleet. There is no significance in the order in which the following management decisions and policies are presented.

Most of them are performed by many organizations. The significance of our implementation is that we do each of them to the Nth degree.

a. We have *one* prime contractor. The programming was done by two subcontractors under supervision and management of the RCA, the prime. Moreover, RCA is responsible for building and integrating the entire weapon system which greatly simplifies things. (The development of the MK 26 launcher and SM-2 missile are by other primes, but RCA is responsible for integration. Besides, these do not involve the computer programming under discussion.)
b. We use only unit computers. Four computers are used in operation of the first engineering development model, two of which are required to operate the radar. Many have questioned why we did not use multiprocessing since a great deal of information is passed from computer to computer during every second of time. We felt in the beginning (and even more so now) that multiprocessing introduces a whole new set of problems. We have found only one pre-AN/UYK-7 tactical military system that uses multiprocessing, though many systems are planning multiprocessing and some are having difficulties.

We are not against multiprocessing. We are planning memory sharing for our DG operation and will leave open an option to go to multiprocessing. However we feel that for tactical systems the multiprocessing problems are very real and very subtle. We would like for others to solve them. We have our hands full (and have been successful) making our computer programs for AEGIS without taking on the task of tactical multiprocessing.

c. Programming-wise, we divided AEGIS into five autonomous computer programs. Instead of having a single computer system developed by a single group of people we have five groups, relatively independent, each writing a separate computer program. The programs are:

    Radar Control
    Weapons Control
    Control Command
    ATEP (the executive)
    ORTS (Operations Readiness Test System) which resides in the radar and weapons control computers.

Each of these was developed separately from documentation through test and integration. Each was monitored and accepted by a different group in the Navy. Reducing the total programming effort to five smaller homogeneous pieces has made it possible to "eat the elephant" even without indigestion.

d. We have a Computer Program Integration Document (CPID). In order for the five antonomous programs to mesh several hundred times per second, all the interfaces must be clearly defined and enforced. The CPID is a working document that does just that. Each message from one segment to another is defined to the bit level and the manager of each segment must sign agreement. Without this document, integration would have been essentially impossible. CPID has been one of the most successful documents published in AEGIS. It is a living workers' document that at one period was changing almost every week. However, every change must and does carry the signature of both parties to the interface.

The Navy can take no credit for CPID. It was generated by RCA when they saw the need early in the program. It is not described in the contract but is informally given to the Navy. It will be required in the next phase of AEGIS.

e. We developed the AEGIS Tactical Executive Program (ATEP) as the common executive for use in all the computers. ATEP is an extremely versatile executive able to handle the requirements of the radar, the weapon system, and the control command system, and it handles all of them very well. It has also been very helpful to have only the one executive. Priorities originating in one segment are carried across segment boundaries. Intersegment interfaces are greatly simplified. Programmers can be shifted from segment to segment with changing work loads because they all interface with the same executive.
f. We placed the equipment and computer programs (hardware and software) under the same man in each segment. The computer programs that make the radar work are just as essential to the radar system as the phase shifters. Therefore, the man in charge of the radar development must also be the one in charge of development of the computer programs that control it. This is also true for the weapons segment, the Control Command segment and the ORTS segment. Each segment has its own group of computer programmers that operate under the same overall ground rules and control. Each segment manager is responsible for the entire development of those programs from requirements through integration. Interfaces between computer program and equipment are worked out at a very low level. Changes in interface can be made without making a federal case. Interpretations of interface descriptions are ironed out at the working level. The normal fingerpointing between programmers and equipment builders is eliminated and integration is much, much simpler.
g. We have a Computer Program Development Plan. This has not been a one-time publication, but has been a living document reissued as the occasion arose or our procedures developed. It has not been just a document for show (though we are proud of it) but a document that is used by the Navy and the prime.
h. We use the "build a little, test a little, integrate a little" procedure. Instead of writing the whole program for a segment and then testing and then integrating

with the equipment, we do it in small pieces. As small portions (modules and usually sub-modules) are completed they are tested. But it does not stop there. These small portions of computer programs are integrated with the equipment. In fact, in the radar, weapons, and ORTS segments, the order in which pieces of equipment and computer programs were developed was coordinated so that the program could be integrated with the equipment as each was being developed. This was possible because both were under the same lower level management. This was also done across segment lines: program modules from SPY-1 and program modules from Control Command were passing messages computer to computer almost a year before either program was complete. These were informal tests and integrations. However, this has paid off exceedingly well in the formal test and integration.

i. We have comprehensive test plans. We realize that no matter what the requirements are, you get nothing beyond what is required to pass tests. The testing of computer programs is done just as is the testing of equipment. We have a test plan document to describe how the test plans are written, how the test procedures are written, and how the tests are to be conducted. With each sub-test, pass/fail limits are specified. The Navy program experts responsible for each test witness that test and verify the results to the bit level. When short-falls develop the program is corrected and all related tests are done over.

Another important thing about acceptance tests: all of them are conditional. The contract requires that the prime *make the AEGIS system operate.* Conceptually it is possible for RCA to pass all the individual tests and the system not operate. (Several years ago this happened in another service.) In which case the contractor would have to fix the system so it would work.

j. The Navy project office has computer experts. The manager of each of the five segments with computer programming either is a computer expert or has a consultant whom he freely uses. These experts take part in all reviews (formal and informal) and oversee the contractor down to code level. The AEGIS Project Manager has a computer expert on his personal staff. The duties of these experts are not only fiscal and managerial but principally technical.

k. The Navy computer program experts visit RCA, CSC and Raytheon. Since the beginning of the contract they have averaged spending more than one working day in 8 at the contractor's facility. They know the managers and chief programmers. They follow the schedule and fiscal operations. They witness all the demonstrations and tests to the bit level. They reviewed the performance and design specifications.

l. Additionally we have computer program experts in the Technical Representative Office. This Technical Representative Office is an extension of the Navy Project Office in the prime contractor's building. Both this office and the Project Office have a Navy expert for each of the five programming segments. Hence there is daily contact. The Technical Representative Office is in addition to the normal Defense Contract Administration Services Office.

m. We held thorough design reviews. The Preliminary Design Reviews (PDR) and Critical Design Reviews (CDR) were "knock down—drag out, bloody" affairs that took months. Every paragraph in every specification was questioned in detail. The Navy participated at the same depth as the contractor. Eventually a step-by-step procedure for CDR's was worked out and published as guidance to all. Some of the specifications went through several iterations before acceptance. It was a very painful experience for the Navy, the prime, and the subs; but it paid off. We got excellent, detailed specifications that we all understood and interpreted the same way.

n. Naval uniformed personnel operate the computers. They are given several hours each week to sit at the consoles and operate the computers as they control the AEGIS equipment. They conduct all the demonstrations for the Navy Project Office and visitors.

This has assured us the console procedures and their computer programs follow standard Navy practice and that they are adequately described so that prospective users can actually use them. Such personnel have been most helpful in keeping AEGIS operating procedures in the real, practical world.

o. We have made great use of simulators. The Weapons Control Segment has a simulator for the interface of the radar segment and another for the Control Command segment so that the Weapon program can operate as if the other segments are present and operating. Likewise the radar and C C segments simulate each other and the weapons segment. There are also simulations of the launcher, the illuminator, a ship's motion simulator, etc. These simulator computer programs are all checked out and authenticated. They usually (but not always) run in another computer. Programs are first tested against simulators before being tested with equipment or other programs.

p. We have realistic scenarios for test and demonstration. These have enabled the Navy to gain confidence, besides being very useful debugging tools.

q. We have very rigid configuration control of computer programs. Master tapes are kept in two places under lock and key. Changes in computer program are made only after very elaborate procedures. Since the programming is done by subcontractors, each change is separately authorized by the contracting officer at RCA. Current "father," and "grandfather" master tapes are maintained in the library.

r. The Navy placed the computer equipment under configuration control. The computers and their peripherals were procured over a period of years. "Minor" manufacturing changes have been such that computer programs would not have been interchangable except for the rigid configuration control of the project office. Likewise, the maintenance would have been different in the older and newer computers.

s. The computer programming has received topside attention. The head of the AEGIS office has taken personal interest in the computer programs. "The computer program is as integral a part of the system as is the equipment." The computer program development in each segment receives full attention of top management and is discussed at (essentially) every Navy review. Problems that arise in computer programming go to the top for solution just as easily as those in equipment design. This support to the Navy personnel responsible for these programs has been invaluable.

CONCLUSION

We are quite sure that by carrying out these management policies we have:

a. reduced the high risk of computer program development.
b. had fewer than anticipated difficulties with integration (e.g., the radar demonstrated computer controlled tracking capability in half the anticipated time).
c. held down the cost of program development (though aggravated by problems with a new computer).

Therefore to others faced with development of large tactical computer programs we recommend adoption of these management policies.

An optimal pollution surveillance schedule generating system (OPGENS)*

by LYNN J. McKELL

University of Minnesota
Minneapolis, Minnesota

and

GORDON P. WRIGHT and DAVID G. OLSON

Purdue University
Lafayette, Indiana

INTRODUCTION

In recent years there has been a general increase in concern for the quality of the environment.[1,7] While this concern has been manifest in many different ways, considerable attention has focused in the area of water pollution because of the following factors:

(1) A few shipping accidents have caused spectacular local threats to the environment.
(2) Leaks in offshore oil well drilling operations have received widespread news coverage.
(3) A rising demand for petroleum products throughout the world complicated by political considerations has produced a corresponding expansion in tanker shipping operations and in offshore oil operations, both of which increase the potential threat of pollution to the ocean environment.

These factors combined with the increasing public concern has precipitated substantial activity in the area of pollution detection and deterrence. Consistent with these efforts the United States Coast Guard is establishing a surveillance system that will detect and track spillage of oil and other hazardous materials, and monitor changes in pollution levels. The basic system of surveillance within the Coast Guard relies on the use of Coast Guard shore units, harbor patrol craft, and helicopters. Currently an effort to improve this system is being initiated by the use of Grumman HU-16E aircraft equipped with infrared and ultra-violet sensors. The addition of these aircraft represents an effort to use advanced technology to improve the surveillance program.

In this paper procedures are developed for scheduling the use of fixed wing aircraft in pollution surveillance patrol efforts. The scheduling problem of concern is a special case of the more general surveillance scheduling problem which need not be defined in the context of environmental protection. Within the jurisdiction of the Coast Guard are a number of surveillance activities which are also subject to similar constraints and which have similar objectives. Examples of such activities are iceberg patrols, fishery patrols, ocean dumping patrols, and some types of search and rescue patrols. Any of these surveillance activities could be analysed using the procedures presented.

The concepts developed here may also be applied to non-Coast Guard activities. For example, the scheduling of border patrols and United States Customs surveillance are two other potential applications.

POLLUTION DETECTION AND PREVENTION SYSTEM

In support of the surveillance activities to be performed by the HU-16E aircraft, research at Purdue University has focused on the development of a Pollution Detection and Prevention System (PDAPS).

The PDAPS research effort has been divided into two basic phases. In Phase I a major analysis was conducted to examine the source, location, size and type of reported spill incidents. In addition, statistics on tanker and barge traffic as well as tonnages of petroleum products transported over U.S. waters were analyzed and regression models were developed to test them as pollution incident forecast models.

A previous study by the Dillingham Corporation[5] examined a selected number of specific petroleum spill incidents and reported on their effect and various clean up techniques. However, little attention was directed at detection or prevention efforts, and the conclusions drawn with respect to pollution prediction are valid only in the context of the few incidents examined in the Dillingham study.

In contrast, PDAPS examines a much larger data base and

* This research was supported by a contract from the Office of Marine Environment and Systems, United States Coast Guard, Contract Number DOT-CG-23308-A.

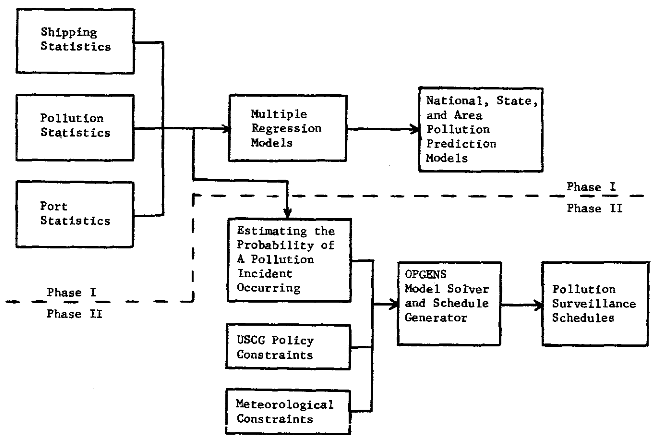

Shipping Statistics
Pollution Statistics
Port Statistics
Multiple Regression Models
National, State, and Area Pollution Prediction Models
Phase I
Phase II
Estimating the Probability of A Pollution Incident Occurring
USCG Policy Constraints
Meteorological Constraints
OPGENS Model Solver and Schedule Generator
Pollution Surveillance Schedules
Phase I
Phase II

Figure 1—Pollution detection and prevention system (PDAPS)

focuses explicitly on the prediction, detection and prevention problems.

The Coast Guard administratively is divided into several large geographic districts. The results of Phase I of PDAPS were used to define geographic sectors in each of these Coast Guard districts where pollution incidents or potential polluting activities are likely to occur. These sectors are defined sufficiently small so that the HU-16E aircraft can patrol any sector in a reasonable amount of time (say 30 minutes or less).

The sectors are rectangular in shape so that patrol patterns can be easily specified and coverage factors and patrol times can be easily calculated. The exceptions to this are sectors defined for portions of the inland waterway system which may not conveniently follow a straight line path. The sectors defined by the PDAPS Phase I effort are generally found to be either areas of concentrated offshore oil well drilling operations or areas of high density tanker and barge movement such as established shipping lanes and channels, bay areas, port areas, and dock areas.

The second phase of PDAPS involves the development of surveillance scheduling models which use some of the information from Phase I as inputs and which produce as an output the flight schedules suitable for use in directing the surveillance patrol efforts of the aircraft to the sectors as specified in Phase I.

The aircraft pollution flight schedules provided by PDAPS for the Coast Guard are generated with two objectives in mind: First, the pollution flight schedules should maximize the expected number of *pollution incidents detected*. Second, over time these schedules should be random so as to have a preventive effect (deterrence) on intentional polluters.

A general diagram of the Pollution Detection and Prevention System is shown in Figure 1. From this figure it is seen that Phase I is a major effort in collecting and analyzing data on activities which may be related to various forms of pollution incidents. In Phase II the specific needs of the pollution surveillance system are examined and the constraints imposed by the implementing organization are considered. Information generated by the Phase I analysis is used as input to Phase II, and the output from the Phase II effort is a set of pollution surveillance schedules.

It should be emphasized that while the surveillance schedules are designed for use specifically by the HU-16E aircraft, their use need not be restricted to this application. The schedules are general enough that they could also be used, for example, by helicopters or other fixed wing aircraft in directing visual surveillance efforts. The only fundamental difference between surveillance using these aircraft and surveillance using the HU-16E is in the nature of the sensing mechanism. Within the context of PDAPS it is possible to compensate for this difference by changing only one parameter (sensor sweep width) in order to design schedules for specific use by these other aircraft.

OPTIMIZATION MODELS

The techniques used in the surveillance patrol models are based on work reported in the Markovian decision process literature. Specifically, the optimization models represent an extension and application of work done by Derman and Klein on the stochastic traveling salesman problem.[4] For a given set of input parameters the output of the patrol models is a set of decision rules which give the optimal probabilities for going from one pollution sector to another pollution sector. The optimization models determine these optimal probabilities in a manner that maximizes the expected number of detections of oil and hazardous material spills given that there is a constraint on the number of hours available for a pollution patrol mission.

In presenting the various models, reference will be made both to sectors and to states. While the difference between these two terms is not difficult to understand, it is important that the relationship between sectors and states be made explicit. The defined pollution sectors are geographic areas corresponding to ports, river and canal segments, segments of shipping lanes, bays, docks, and areas denoting offshore oil producing operations. Sectors are defined on the basis of PDAPS Phase I results describing the actual occurrence of pollution incidents or the potential occurrence of pollution incidents as predicted from shipping statistics.

Prior to beginning a patrol mission, one of the parameters which must be specified is the amount of flight time to be allocated for that particular surveillance patrol effort. This mission flight time, M, is a constraint on the surveillance activity, in that no pollution surveillance schedule should be designed which requires more than M time units to complete. As a route defined by a particular schedule is covered in a surveillance mission, the mission time resource is expended or used in two basic patrol activities. Specifically, some time is required in transiting from one sector to another sector and time is required to perform a search of a given sector according to a specific patrol pattern. Upon completion of the patrol pattern the aircraft is at one of the sector end points prepared for transit to the next sector.

In these models it is assumed that travel to a new sector is initiated only if there is adequate time to complete the transit and a search pattern with enough time subsequently remaining for travel to the terminal air base.

Thus a critical decision point is reached at the conclusion of each search pattern just prior to the next move. In these models a state is identified and created for every feasible critical decision point as described above. The state definition consists of a three-tuple which uniquely defines the sector, the sector end point, and the amount of time yet remaining in the mission. Let $s(k, e, b)$ denote a state where k refers to a sector number, e is an end point identifier, and b refers to the amount of time remaining in the mission. It is important to realize that for any particular starting point (air base) and mission flight time (M), only a finite number of states are possible because a measurable amount of time is expanded for the travel and search effort in reaching any particular critical decision point. Several observations can be made about states. First note that any particular sector can be identified with many different states; however, the reverse is not true. Second, whereas a sector has only spatial dimensions, a state has a time dimension in addition to spatial dimensions so that a state represents a point in time with respect to the particular mission as well as a geographical location. In OPGENS each defined state is assigned a unique cardinal number which has no direct correlation with any element or elements of the three-tuple state identifier except through this arbitrary assignment.

Suppose the current critical decision point corresponds to state i, then for each feasible critical decision point directly reachable in one move from the current position there corresponds a state. The set of all such feasible states directly reachable in one move from the current state i is called the successor set of i, denoted S_i.

Note that in referring to a state transition a special kind of "motion" is implied. Such a transition implies not just a travel effort but also a specific search pattern activity in the sector corresponding to the terminal state. It is also important to realize that not all state transitions are feasible. Rather a direct transition from state i to state j is feasible only if $j \in S_i$. Observe that under conceivable, albeit unusual, circumstances it is possible that $j \in S_i$ and $j \in S_k$ for $k \neq i$. Also important is the fact that even if this condition occurs, the corresponding state transitions may imply different search activities.

PATROL PATTERN DESCRIPTION

Each state transition implies a specific patrol activity over the sector corresponding to the terminal state. Depending upon the width of the pollution sector and the altitude of the patrol aircraft, a number of different surveillance patrol patterns may be defined for each geographic sector. Figure 2 shows a number of alternative patrol patterns which OPGENS may evaluate for a particular sector. The patterns defined in PDAPS and considered by OPGENS are referred to in the search theory literature[6] as Parallel Track-Single Unit patrol patterns. The fraction of the sector area covered by each

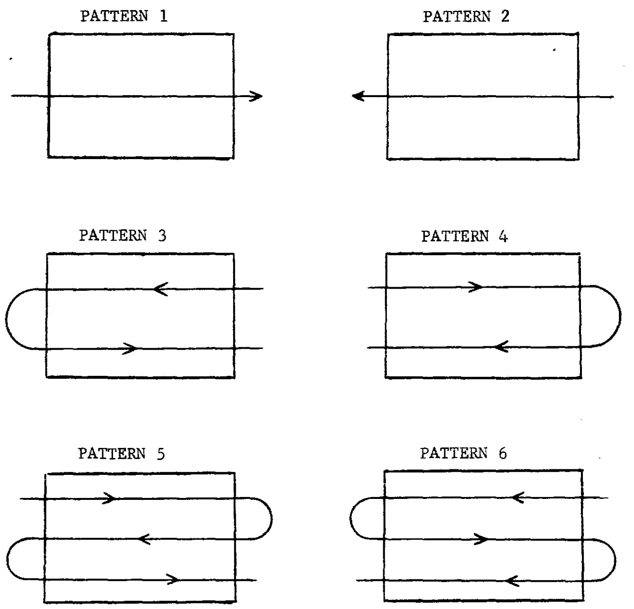


Figure 2—Six different flight patterns for a geographical pollution sector

pattern is an estimate of the conditional probability of a detection given that a spill is present in the sector during the surveillance search effort. This quantity is denoted as the coverage factor, CF_{ij}, associated with the search activity implied by the transition from state i to state j. Estimating the amount of flight time between geographic sectors and the amount of time required to fly a particular pattern is based on the appropriate ground speed at a given altitude for a good sensor performance. Such factors as different wind speeds and wind directions as well as the amount of time spent on detections and false alarm can cause the actual flight time spent in a pattern to vary from this estimate. While flight times are treated as known quantities, they really represent the best estimate of their expected values.

As characterized, there is a benefit in terms of the expected probability of detecting an oil spill for a transition from one state to another state. There is also a cost incurred for this transition. The cost is the sum of the expected flight time to go from one geographic sector to another plus the expected time spent in flying the particular pattern which effects the transition to the successor state. The value of the remaining flight time b in the state $i = s(k, e, b)$ restricts the number of states in the set S_i.

BASIC MODEL FORMULATION

Let S denote the set of all states to be considered for a particular pollution patrol mission. Let 0 denote the home air station. Assume S contains N states and let S_i denote the set of feasible successor states to state i $(i = 0, 1, \ldots, N)$. (Note that $S = S_0 \cup S_1 \cup \cdots \cup S_N$.)

Let M denote the total flight time available for the pollution patrol mission. Let τ_{ij} denote the distance (in minutes) for the aircraft to effect the transition from the exit point of state i to the exit point of state j. $\tau_{ij} = TT_{ij} + PT_{ij}$ where TT_{ij} is the distance (in minutes) from the exit point of state i to the entry point of the sector associated with state j and PT_{ij} is the time to patrol the terminal sector according to the pattern associated with the transition to state j from state i.

Let Pd_{ij} denote the probability of detecting a pollution incident while patrolling the sector-pattern combination associated with the transition from state i to state j. At present $Pd_{ij} = (CF_{ij})(PC_{ij})$. Where CF_{ij} is the coverage factor associated with the transition, PC_{ij} is determined by taking the ratio of the total number of pollution incidents which occurred in the geographical sector associated with the transition to the total number of pollution incidents which occurred over all pollution sectors associated with all the states in S. This is used in the oil producing sectors in the Gulf. Otherwise, the fraction of all bulk cargoes of petroleum products traveling through the sector is used. Admittedly, it would be more desirable to use the a priori probability $(AP_{ij}(t))$ of a "pollution incident in progress" at time t, (the time of surveillance), instead of PC_{ij}. However, the data, at present, is not available for such estimation. Hence PC_{ij} is used as a surrogate measure of $AP_{ij}(t)$.

Let X_n denote the nth state transition upon leaving state 0. Note $X_1 = (0, i)$, $i \in S_0$ and $X_T = (j, 0)$, $\{S_j\} = 0$, where the Tth transition is the last transition in a particular tour schedule. Next let $C_{X_n} = 1$ if a pollution incident is detected during the surveillance patrol associated with the nth transition. Let $C_{X_n} = 0$ if no pollution incident is detected. Then

$$E[C_{X_n} \mid X_n = (i, j)] = 1 \cdot Pd_{ij} = Pd_{ij} \tag{1}$$

for all $i \in S$ and $j \in S_i$.

For any particular state i suppose there are m states $(s_1, \ldots, s_m)$ in S_i. Thus from state i there are m possible actions which can be taken. Let $A_n = (i, j)$ denote an effort to move from state i to state j ($j \in S_i$) upon completion of X_n. We allow A_n to be chosen randomly by letting

$$D_{ia} = P(A_n = (i, a) \mid X_n = (h, i))\, i \in S \quad \text{and} \quad a \in S_i \qquad i = 0, 1, \ldots, N \tag{2}$$

When the model is in state i, D_{ia} represents a probability distribution over the set of possible actions defined by the successor set, S_i. These D_{ia}'s are the decisions to be determined by the model and satisfy the normal conditions implied by a probability distribution. That is

$$0 \leq D_{ia} \leq 1, \quad \text{for} \quad a \in S_i \quad \text{and} \quad \sum_{a \in S_i} D_{ia} = 1 \tag{3}$$

Next, certain laws of aircraft movement which guarantee the introduction of randomness into the flight schedules are defined.

Let $q_{ij}(a) = P\{X_{n+1} = (i, j) \mid X_n = (h, i), A_n = (i, a)\}$

$$= \begin{cases} 0 & \text{if } j \notin S_i \\ 1 - \sum\limits_{j \in S_i - \{a\}} f(i, j, a) & \text{if } a = j \text{ and } a, j \in S_i \\ f(i, j, a) & \text{if } a \neq j \text{ and } a, j \in S_i \end{cases} \tag{4}$$

where $f(i, j, a)$ is a function which introduces uncertainty into the scheduling model. If the decision is made to go from state i to state a in S_i, $D_{ia} = 1$, then the movement is made with probability $(1 - \sum_{j \in S_i - \{a\}} f(i, j, a))$ instead of with probability 1.0 (a deterministic decision). Upon leaving state i the aircraft goes to state j with probability $f(i, j, a)$ ($j \neq a$; $a, j \in S_i$). In order that the set of all feasible states, S, be irreducible we require that $f(i, j, a) > 0$.

For the models examined here, let

$$f(i, j, a) = \begin{cases} \dfrac{\epsilon}{N(S_i) - 1} & \text{if} \quad N(S_i) > 1 \\ 0 & \text{if} \quad N(S_i) - 1 \end{cases} \tag{5}$$

where ϵ is a positive number $0 < \epsilon \leq 1$, and $N(S_i)$ is the number of states in the successor set S_i. This definition of $f(i, j, a)$ is called the UNIFORM policy because ϵ is uniformly distributed. The value of ϵ is an input parameter to the model.

Once the optimal decisions (denoted D_{ia}^0) are determined for all $a \in S_i$ and $i \in S$, then the optimal Markov transition probabilities are given by

$$P_{ij}^0 = \sum_{a \in S_i} q_{ij}(a) D_{ia}^0 \quad \text{for all } i \text{ and } j \in S \tag{6}$$

Note that $P_{ij}^0 = 0$ for all $j \notin S_i$.

Let $R = \{D_{ia}\}_{i \in S, a \in S_i}$ denote any set of decisions. Then the scheduling model can be stated as

MODEL I: Find R to

$$\text{MAXIMIZE} \quad E_R\left(\sum_{n=1}^{T} C_{X_n} \mid X_0 = 0\right) \tag{7}$$

subject to

$$E_R\left(\sum_{n=1}^{T} \tau_{X_n} \mid X_0 = 0\right) \leq M \tag{8}$$

where $X_0 = 0$ denotes the initial state is 0, and

T is the number of transitions in a schedule, and
M is the amount of time allocated for the patrol mission.

The objective function is the expected number of pollution incidents detected while searching the sectors associated with the T states in the schedule defined by R. Equation (8) is the budget constraint where τ_{X_n} is the time (in minutes) to effect the nth transition.

MODEL FOR MAXIMIZING EXPECTED BENEFIT PER PATROL MISSION

Let

$$x_{ia} = \pi_i D_{ia} \qquad i \in S, a \in S_i \tag{9}$$

where π_i's ($i \in S$) are the steady state probabilities associated with the P_{ij}'s defined in equation (6).

Then we have

$$\pi_i = \sum_{j \in S_i} x_{ij} \tag{10}$$

As discussed previously Pd_{ij} is the expected benefit for performing the search implied by the transition from state i

to state j. This benefit is obtained only if we are in state i and make the decision to go to state j. During any arbitrary mission the expected number of visits to state i is given by π_i/π_0. This stems from the fact that the mission must begin at state 0. Also, we know that P_{ij} is the conditional probability of going to state j given that the process is in state i.

Therefore the objective function given in (7) can be reformulated as

$$\text{Find } P_{ij}(i\in S, j\in S_i) \text{ to}$$

$$\text{MAXIMIZE} \quad \sum_{i\in S}\sum_{j\in S_i}(\pi_i/\pi_0)P_{ij}Pd_{ij} \tag{11}$$

Substituting the expression for P_{ij} from (6) yields

$$\text{MAXIMIZE} \quad \sum_{i\in S}\sum_{j\in S_i}(\pi_i/\pi_0)\sum_{a\in S_i}q_{ij}(a)D_{ia}Pd_{ij}$$

$$= \text{MAXIMIZE} \ (1/\pi_0)\sum_{i\in S}\sum_{j\in S_i}\sum_{a\in S_i}q_{ij}(a)\pi_iD_{ia}Pd_{ij} \tag{12}$$

By substituting (9) and (10) into (12) the objective function becomes

$$\text{Find } \{x_{ia}\}_{i\in S, a\in S_i} \text{ to}$$

$$\text{MAXIMIZE} \quad \Big(\sum_{i\in S}\sum_{j\in S_i}\sum_{a\in S_i}q_{ij}(a)x_{ia}Pd_{ij}\Big)\Big/\sum_{j\in S_0}x_{0j} \tag{13}$$

The budget constraint (2) can be restated as

$$\sum_{i\in S}\sum_{j\in S_i}(\pi_i/\pi_0)\cdot D_{ij}\tau_{ij}\leqq M \tag{14}$$

where τ_{ij} is the time for the transition from state i to state j and M is the amount of time allocated for the patrol mission. Multiplying both sides of (14) by π_0 and substituting in the expressions for π_i's and D_{ia}'s from (9) and (10), then collecting terms results in the budget constraint

$$\sum_{j\in S_0}x_{0j}(\tau_{0j}-M)+\sum_{i\in S-\{0\}}\sum_{j\in S_i}x_{ij}\tau_{ij}\leqq 0 \tag{15}$$

To guarantee that the steady state probabilities (π_i, $i\in S$) associated with the transition probability matrix $P=(P_{ij})$ are correct the following Markov conditions must also be satisfied

$$\sum_{i\in S}\pi_i=1 \tag{16}$$

$$\pi_j=\sum_{i\in S}\pi_iP_{ij} \tag{17}$$

By substituting (10) into (16) we obtain the constraint

$$\sum_{i\in S}\sum_{j\in S_i}x_{ij}=1 \tag{18}$$

Substituting (6) and (10) into (17) yields

$$\sum_{i\in S}\sum_{a\in S_i}x_{ia}(\delta_{ij}-q_{ij}(a))=0 \tag{19}$$

where δ_{ij} is the Kronecker δ defined by

$$\delta_{ij}=\begin{cases}1 & \text{if } i=j\\ 0 & \text{if } i\neq j\end{cases}$$

Equation (19) must hold for all $j\in S$.

The constraints given by Equations (18) and (19) together with the budget constraint (15) and the objective function (13) constitute a restatement of Model I known as Model II.

The objective function of Model II is a linear fractional functional in the decision variables x_{ij}. The first and second constraints given by (15) and (18) both consist of a single equation. However, the third constraint (19) must be defined for every state $j\in S$. Thus it will result in a separate equation for each state found in the model.

SOLUTION TECHNIQUE FOR MODEL II

As mentioned in the previous section, Model II has a fractional objective function. Problems with this structure were first solved by Charnes and Cooper.[2] A method for obtaining a solution to Model II is now given. This method involves a transformation of variables and the addition of one constraint to yield a linear programming model which can be solved using well-known techniques.

Let

$$Z_{ij}=x_{ij}/\sum_{k\in S_0}x_{0k} \tag{20}$$

for $i\in S$ and $j\in S_i$, and let

$$Z=1/\sum_{k\in S_0}x_{0k} \tag{21}$$

Then by substituting (20) into (13), the Model II objective function, a linear objective function in the variables Z_{ij} is obtained:

$$\text{Find } \{Z_{ij}\} \ i\in S, j\in S_i \text{ to}$$

$$\text{MAXIMIZE} \quad \sum_{i\in S}\sum_{j\in S_i}\sum_{a\in S_i}q_{ij}(a)Pd_{ij}Z_{ia} \tag{22}$$

The three constraints (15), (18), and (19) can then be put in a form compatible with (22) by dividing each of the three equations by $\sum_{k\in S_0}x_{0k}$.

Performing this division on (19) and substituting (20) gives a revised budget constraint

$$\sum_{j\in S_0}Z_{0j}(\tau_{0j}-M)+\sum_{i\in S-\{0\}}\sum_{j\in S_i}Z_{ij}\tau_{ij}\leq 0 \tag{23}$$

Performing the division on Equation (18) then substituting (20) and (21) and collecting terms yields

$$\sum_{i\in S}\sum_{j\in S_i}Z_{ij}-Z=0 \tag{24}$$

A similar division and substitution in (19) gives the revised third set of constraints

$$\sum_{i\in S}\sum_{a\in S_i}Z_{ia}(\delta_{ij}-q_{ij}(a))=0 \quad \text{for all} \quad j\in S \tag{25}$$

Finally, note that the z_{ij}'s must satisfy the following condition

$$\sum_{k\in S_0}Z_{0k}=1 \tag{26}$$

if the transformation is to hold true. This last equation is a fourth constraint.

Together the set of constraint equations (23) to (26) with the objective function (22) constitute a linear programming problem denoted as Model III.

Solving Model III gives an optimal solution Z^0 and $\{Z_{ij}{}^0\}$, $i\in S$, $j\in S_i$. From this the optimal solution to Model II is given by

$$x_{ij}{}^0=Z_{ij}{}^0/Z^0 \qquad i\in S, j\in S_i \tag{27}$$

and

$$D_{ia}=x_{ia}{}^0/\sum_{j\in S_i} x_{ij}{}^0 \qquad i\in S, j\in S_i \tag{28}$$

Note, however, that

$$D_{ij}=\frac{x_{ij}}{\sum\limits_{j\in S_i} x_{ij}}=\frac{x_{ij}/\sum x_{0k}}{\sum\limits_{j\in S_i} x_{ij}/\sum x_{0k}}=\frac{Z_{ij}}{\sum\limits_{j\in S_i} Z_{ij}} \tag{29}$$

Therefore the x_{ij}'s need never be calculated explicitly. Rather the solution required ($D_{ij}{}^0$'s) can be obtained directly from the results of Model III.

Then the optimal transition probability matrix, $P^0=(P_{ij}{}^0)$ is given by (30)

$$P_{ij}{}^0=\sum_{a\in S_i} q_{ij}(a)D_{ia}{}^0 \qquad i\in S, j\in S_i \tag{30}$$

PDAPS also defines a Fourth model very similar to MODEL III except the objective function is based on maximizing the expected benefit per state transition. OPGENS has the capability of structuring the problem either way as the user specifies.

STATE GENERATION METHODS

As was mentioned in the state description, there are a finite number of states which are feasible for any particular patrol mission. The purpose of this section is to give a method for generating all of the states to be used in the model.

Recall that a state, denoted by $s(k, e, b)$, is formally defined by three parameters: k = the sector identifier, e = exit point identifier, b = amount of time remaining before the allotted mission patrol time has expired. For any sector there are virtually an infinite number of points on the perimeter which could conceivably be points of entry or exit. To avoid the complications inherent in considering a large number of physical entry-exit points the number of such points defined for each sector has been limited to two, one centered at each end of the sector. These are located such that the bearing from one point to the other defines the axis which the aircraft will use in flying the patrol patterns over a sector.

The state generation procedure consists of enumerating all possible paths from the starting air base to the terminating air base which do not violate the specified patrol mission time constraint. Upon arrival at the terminating air base it is assumed that refueling occurs and that complete renewal is effected instantaneously. In a formal sense this implies that by assuming the patrol missions will go on indefinitely the following conditions hold:

(1) any state is accessible from any other state (though not necessarily during the same patrol mission),
(2) all states communicate with all other states, and
(3) therefore, the set of all states, S, constitutes a single Markov class.

The following definitions will be helpful in presenting the state generation algorithm.

(1) For every state $j\in S_i$ ($i\in S$) the state i is called a predecessor state to state j.
(2) Beginning at the air base the set of all feasible successors directly reachable in one transition from the air base is called the first successor generation set and is denoted by $G(1)$. So $G(1)=S_0$. Then for $k>1$ $G(k)$ is defined as

$$G(k)=\bigcup_{j\in G(k-1)} S_j$$

and is called the kth successor generation set.
(3) For any state i, j is a feasible successor state to i only if $j\in S_i$.

We require that no state h can be included in S_i if the sector associated with h is the same sector as that associated with state i. This prevents repeated visits to the same sector without an intervening visit to at least one other sector.

We now present an algorithm for generating all feasible states to be used in a model for a specific set of input parameters defining the surveillance patrol mission.

Procedure 1A:

1. Beginning at the air base with a specified altitude, speed, and patrol mission time, set the initial state (0) corresponding to the air base.
2. By sequentially scanning the list of all sectors, generate and store each feasible state, thus creating S_0 and $G(1)$. Set the successor generation set pointer $k=1$.
3. Increment $k=k+1$
4. Sequentially examine each state $i\in G(k-1)$. For every non-terminal state i (that is, non-air base state) generate and store all feasible successor states to i (that is, create S_i). Note from definition 3 that S_i consists either of states which have associated sectors or a single state corresponding to the terminal air base.
5. If there were any non-air base states in $G(k)$ go to 3. Otherwise stop; the set of all feasible states, S, is complete.

Several observations can be made about Procedure 1A and the resulting structure of S. (a) This procedure guarantees that the aircraft will terminate at the destination air base. (b) The aircraft will travel to the air base only if there are no defined sectors which can be patrolled within the allotted time. This precludes the patrol mission from terminating prematurely. (c) It is possible for a state having the same identifiers to be generated more than once in different successor sets. (This, of course, assumes a finite resolution to measurement of time.) (d) Procedure 1A creates the feasible states one successor generation set at a time with each successor set being produced sequentially in the order which its predecessor state is found in the previous successor generation set. (e) Finally observe that the structure of the implied state transitions is in the form of a full tree network with the exception that all paths terminate into a common node (the destination air base).

A comparison of Procedure 1A with the requirements of the model reveals that the principal disadvantage to the procedure lies in the difficulty of identifying the valid state transitions. Therefore the following modification is made to Procedure 1A: after each new state is created, store the identifiers of both the new state and its predecessor state together. Procedure 1A with the addition of this modification will be called Procedure 1B. With Procedure 1B it is a trivial matter to identify both the states and the state transitions for inclusion in the appropriate model.

THE USE OF STANDARD LINEAR PROGRAMMING SYSTEMS

Both of the optimization models (Model III and Model IV) designed for surveillance scheduling have linear objective functions. Hence it is possible to consider many standard computer programs designed to solve such models.

In order for a linear programming computer program to be compatible with the OPGENS concept it should include many of the following desirable features.

1. The data input format should be easy to generate.
2. The program should be capable of handling large problems (at least 1600 rows and 3000 columns).
3. The output should allow for easy extraction of the solution for post-optimality processing.
4. An examination of Models III and IV reveals that there is only one non-zero element in the right hand side column. The system used should be able to handle this unique structure.
5. In anticipation of future modifications to OPGENS it may be desirable if the selected linear programming system would allow for read-in of an improved starting basis.
6. The system should be efficient so that use of the system can be justified cost-wise.

At Purdue University some of the linear programming packages available for users are MPS on IBM 360 equipment; LP90, OPTIMA, RADES, and VIVACE' on a CDC 6000 series computer. Of these the one which was most compatible with OPGENS' requirements was OPTIMA.[3] While OPTIMA has various solution algorithms available, OPGENS uses the option called PRIMAL which is basically a form of the revised simplex (RS) algorithm well-known in the mathematical programming literature. Theoretically, the implemented version of OPTIMA allows for up to 4095 rows and has virtually no column restrictions. The principal drawback to OPTIMA is its inefficiency—it is quite time consuming.

A METHOD FOR SETTING A STARTING BASIS FOR THE OPTIMIZATION MODELS

One of the major disadvantages of using the RS algorithm to obtain globally optimal solutions is that programs which implement the algorithm require extensive computational time. While this is generally true for most standard computer programs, it is particularly true of OPTIMA. One method for increasing the efficiency of the optimization process is to introduce an initial basis which is closer to the optimal basis than the one consisting of artificial and slack variables which is generated by OPTIMA (and most other systems). It can be shown that any optimal solution to Models III or IV will contain in the basis exactly one transition from each state i to some successor state $j(j \in S_i)$.

Therefore, the following heuristic is suggested as a simple, rational rule to use in selecting the transition activities to be included in the basis. Both Models III and IV are maximization models. For ϵ small (say less than .5) it is not unreasonable to expect that the optimal solution basis will include many transition activities which have a high benefit contribution to the objective function. Thus the basis selection rule is: for each state, i, which has only one element, j, in S_i, include that transition, Z_{ij}, in the set of starting basis activities; for each state, i, which has more than one successor element in S_i, select the transition Z_{ik} such that $Pd_{ik} = \text{Max}_j \{Pd_{ij}\}$ for $j \in S_i$. The validity of using this heuristic depends on whether the global optimal solution basis has a close correlation with the results of this one-step look-ahead local search technique.

OTHER SOLUTION ALGORITHMS

Early experience using the RS algorithm to obtain solutions to models III and IV on some of the smallest problems indicated that optimal solutions were very costly. Therefore, concurrent with the development of the optimal pollution surveillance scheduling system, research was conducted to investigate various suboptimization approaches to obtaining a solution. Two of the algorithms preserve the stochastic nature of the model solutions. A third algorithm uses a one step-local search (LS) technique to produce a deterministic set of schedules.

Besides the problem of computer time and costs involved in solving the optimization models, another problem anticipated from early experience focused on the size of the state set. While short flight times force the state set to be reasonably small, the combinatorics involved in the longer flight mission result in state sets and transition matrices sufficiently large as to exceed the capability of present day computers. Therefore a set of techniques was developed to limit the size of the state set generated. These techniques are explained in a later section of this paper.

YQ Algorithm

The procedure for ultimately obtaining the Markov transition probabilities involved the post optimization use of the optimal solution to determine the appropriate D_{ij}^{0}'s which would then be used with the previously defined $q_{ij}(a)$'s to produce the P_{ij}^{0}'s.

The *YQ* algorithm consists of using the basis set derived from the starting basis selection heuristic as the solution basis for use in the "post optimality" processing. The obvious assumption is that the starting basis so derived results in an objective function value sufficiently close to optimality.

While it may appear at first that little is gained by omitting a few optimization iterations, the OPGENS process for deriving such a solution by-passes, in addition, a substantial amount of computing required to prepare input matrices for the OPTIMA system and to extract the *RS* algorithm's solution.

In summary, the system based on the *YQ* algorithm is conceptually very similar to the one based on using the *RS* algorithm except that the method for determining the solution basis does not guarantee optimality.

NQ Algorithm

In contrast to the *YQ* algorithm which still retains the capability of controlling uncertainty through specifying ϵ, the *NQ* algorithm surrenders that control. In the optimization models and the *RS* and *YQ* algorithms the probability distribution defining the state transitions were a function of ϵ. In the *NQ* algorithm these transition probabilities are set according to the following rule.

Let

$$PSUM_i=\sum_{j\in S_i} Pd_{ij} \qquad \text{for all} \quad i\in S$$

then

$$P_{ij}^{0}=Pd_{ij}/PSUM_i \qquad \text{for all} \quad i\in S$$

Thus the transition probability from a given state, i, to any state j in S_i is proportional to the size of the benefit associated with that state transition (relative to the benefits of other feasible transitions from the state). So transitions with larger benefits have a better chance of being selected in the schedule generation phase.

One-Step Local Search (LS) Algorithm

The *RS*, *YQ* and *NQ* algorithms all generate probability distributions which are subsequently sampled using random numbers to produce actual sets of transitions for the schedules. Thus these three algorithms preserve the randomness of the schedules.

In contrast, the *LS* algorithm to be described directly selects transitions to be used in the schedules according to the one-step local search rule already described in the section on basis selection. This generation of schedules is performed according to the following procedure.

1. Generate all states for the first two successor generations according to Procedure 1B.
2. For each state in generation $G(2)$ define a schedule and include as the first two transitions in each schedule the two unique transitions which result in arrival at the corresponding state.
3. For each non-terminated schedule select as a next transition (from the set of feasible state transitions) the one which has the largest immediate benefit.
4. For all states whose feasible successor set is null, terminate the schedule by a transition to the air base.
5. If there are yet remaining any non-terminating schedules go to 3. Otherwise stop.

This algorithm has two principal drawbacks:

1. The schedules are not random.
2. The number of schedules generated is limited to the number of states in the second generation, $G(2)$.

TECHNIQUES FOR RESTRICTING STATE GENERATION

Suppose that state generation for a five-hour problem results in defining approximately 10 successor generations. Suppose further that each state in the first seven successor generations has only 10 successor states, and only one successor state exists for each state beyond the seventh generation. Then such a model would result in $N=\sum_{i=1}^{7} 10^i+3*10^7\cong(4*10^7)$ states and even more state transitions. Unfortunately, OPTIMA (which is generally considered capable of handling fairly 'large' problems) is limited to 4095 rows and (practically) 5000-10,000 columns. Thus the hypothetical problem structure defined above (which was purposefully made conservative) is much too large for solving with OPTIMA (or, for that matter, any other currently existing optimization system). Thus for obtaining practical solutions some techniques must be used to limit the problem size to less than 4000 rows and about 5000 columns. Three techniques were incorporated into the state generation program to accomplish this for the described models.

Technique 1 consists simply of limiting the sector successor set size. For each sector a physical observation was made to determine which other sectors were reasonable candidates for being visited next. In general the policy used was that from a given sector any neighboring sector reachable without flying over another sector should be in the successor set. The method of implementing this technique then is to modify Procedure 1B so that only neighboring sectors are considered when defining successor states.

Technique 2 consists of counting unique states only once in the problem even though the same state may be generated as a successor to many different states in several different successor generations. Before this technique could be implemented, it was necessary to define the degree of accuracy required in measuring time for state definition purposes. Because of the various factors influencing the flight of low speed aircraft it was decided that time measurements to the nearest minute were sufficiently accurate without being absurdly precise.

Thus with Technique 2 the maximum number of states for any problem is limited to NS_{MAX} given by

$$NS_{\text{MAX}} = 2 \times \text{number of sectors} \times \text{number of minutes}$$

So for a five-hour (300 minutes) flight in the Gulf of Mexico (which currently has 105 sectors defined) the maximum number of states which could possibly be generated is

$$NS_{\text{MAX}} = 2 \times 105 \times 300$$
$$= 63{,}000$$

The effect on Procedure 1B of implementing this technique is first to specify the method for measuring time, and second to add the condition that once a state has been processed as a predecessor (by "processed" it is implied that the successor set has been generated), there is no need to ever again process it as a predecessor. A hashing scheme was used to increase the efficiency of this process.

The use of these two techniques resulted in an adequate reduction of the problem size for short duration patrol schedules. However, the problems with longer patrol mission times were still much too large. As a final measure, Technique 3 consists of randomly limiting the size of the successor sets to insure that the problem is solvable. The algorithm which effectively implements this technique within the framework of Procedure 1B is the following:

1. For a given generation, reset the maximum number (m) of successors allowed in a successor set.
2. For a given predecessor state, i, generate all successor states according to Procedure 1B as limited by Techniques 1 and 2.
3. If the number of successor states is less than m allow all of the successors to be in S_i.
4. If the number of successors exceeds m, randomly, select m unique states to be included in S_i.

Thus, by properly setting m, the problem size can be controlled.

The principal purpose of Techniques 1 and 3 is to introduce modifications to Procedure 1B which effectively limit the number of states generated according to some rational basis. These methods were needed not because of any inherent weakness in Procedure 1B, but rather because of problem size considerations.

There is, however, a characteristic exhibited by schedules generated on the basis of Procedure 1B as modified by Techniques 1 and 3 which may be undesirable for some applications. This characteristic is a tendency for sectors scheduled for patrol to be clustered in the close proximity of the originating air base.

This cluster effect is perhaps particularly undesirable in generating schedules which have a different originating and terminating air base. The employment of Techniques 1 and 3 resulted in schedules where most surveillance was conducted near the originating air base, with the final transition of the schedule being a very long direct flight from some sector to the terminating air base.

In order to eliminate this effect Technique 4 was developed with two different rules for two different cases. Case 1 is employed in the single air base problem:

> Case 1 Rule: Sectors visited should be progressively farther away from the originating air base.

Obviously, this rule should be effective during only a portion of the mission; otherwise the final transitions would likely be inordinately long. In order to establish when the rule is effective and when it is not, the average time remaining (ATREM) for all non-terminal states found in the most recent successor generation is calculated. If ATREM is less than some prespecified time (say 40 percent of the total patrol mission time), then the Case 1 rule is not invoked, otherwise it is invoked to limit the successor set prior to the use of Technique 3. Obviously, an escape mechanism must be implemented in the event that there are no feasible states more distant from the air base reachable from the current state. The escape mechanism employed is that if such is the case, then the Case 1 rule is bypassed and is ineffective for that particular predecessor state.

Case 2 is the two air base problem. In this case it is desirable that the schedule direct the aircraft in a general direction toward the terminating air base.

The following rule is employed to accomplish this:

> Case 2 Rule: Sectors visited should be progressively closer to the terminating air base.

Again provision must be made for making this rule ineffective during the latter portion of the mission. The escape mechanism described for Case 1 rule is also employed with the Case 2 rule.

The previously described Case 1 rule and Case 2 rule constitute an effective method for mitigating the clustering effect resulting from use of Techniques 1 and 3. These two rules together are referred to as Technique 4 for modifying Procedure 1B.

It should be noted that the provision to specify a time range when the technique is effective also allows for total bypass of the technique.

THE INPUT DATA BASE

The models developed for scheduling pollution surveillance require that areas of potentially high pollution activity be well defined and that a quantitative measure of this potential pollution be made for each of the defined areas. There are two

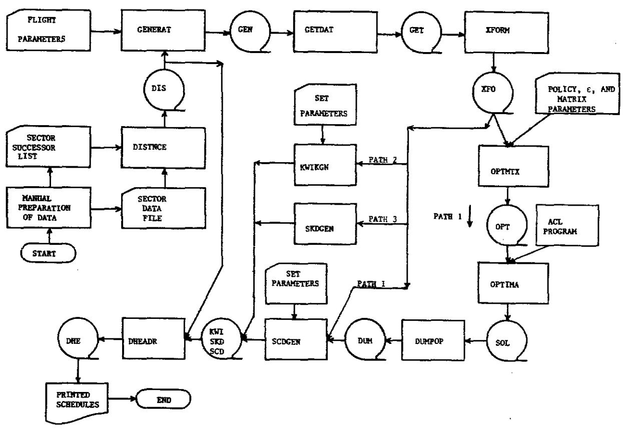


Figure 3—Optimal pollution surveillance schedule generating system

principal activities which produce much of the harmful pollution in the navigable waterway system. These activities both stem primarily from the procurement of petroleum products to satisfy demands.

One of these activities is the shipping of petroleum products in tankers and barges. In the PDAPS analysis consideration of shipping was limited to the routes for ocean-going vessels (tankers), the intercoastal waterway system including relevant rivers and canals, and the Great Lakes shipping routes. The precise location and size of these established shipping routes was obtained by examining standard navigational charts for the areas in question. The shipping lanes were divided into segments which could be patrolled in 30 minutes or less. A segment boundary was also defined where a shipping lane changed directions or intersected with another shipping lane. Using available transportation, shipping, and port statistics the likely routes for shipments were determined and the total tonnage of petroleum material was calculated and tabulated for each sector. The information on sector location, length and width, and the tonnage shipped during the base period (1970 data was used for setting model parameters) was aggregated into a file denoted the sector data file.

The second principal activity considered was that of offshore oil-well drilling operations. In those areas where such activities are substantial maps were used which locate the fields of activity. Rectangular areas were then defined for each field. For these regions historical data is available describing the occurrence and location of reported spill incidents. Unfortunately data on the amount of pollutant spilled was either unavailable or unreliable; therefore an analysis was made to determine only the number of spills (not the amount) that occurred in each of the defined sectors for the base period. The sector location, dimensions, and number of spill incidents was also included in the sector data file. Each record in this file also had a sector identifier which permitted the method for determination of the sector surveillance-benefit weight to be different depending on the pollution activity measure.

In addition to the sector data file, a sector successor file was generated which contained a list of successors which could be reached easily from the current sector. This file was constructed by examining the navigation charts to determine which were neighboring sectors. This file listed up to nineteen successors for each sector.

OPGENS

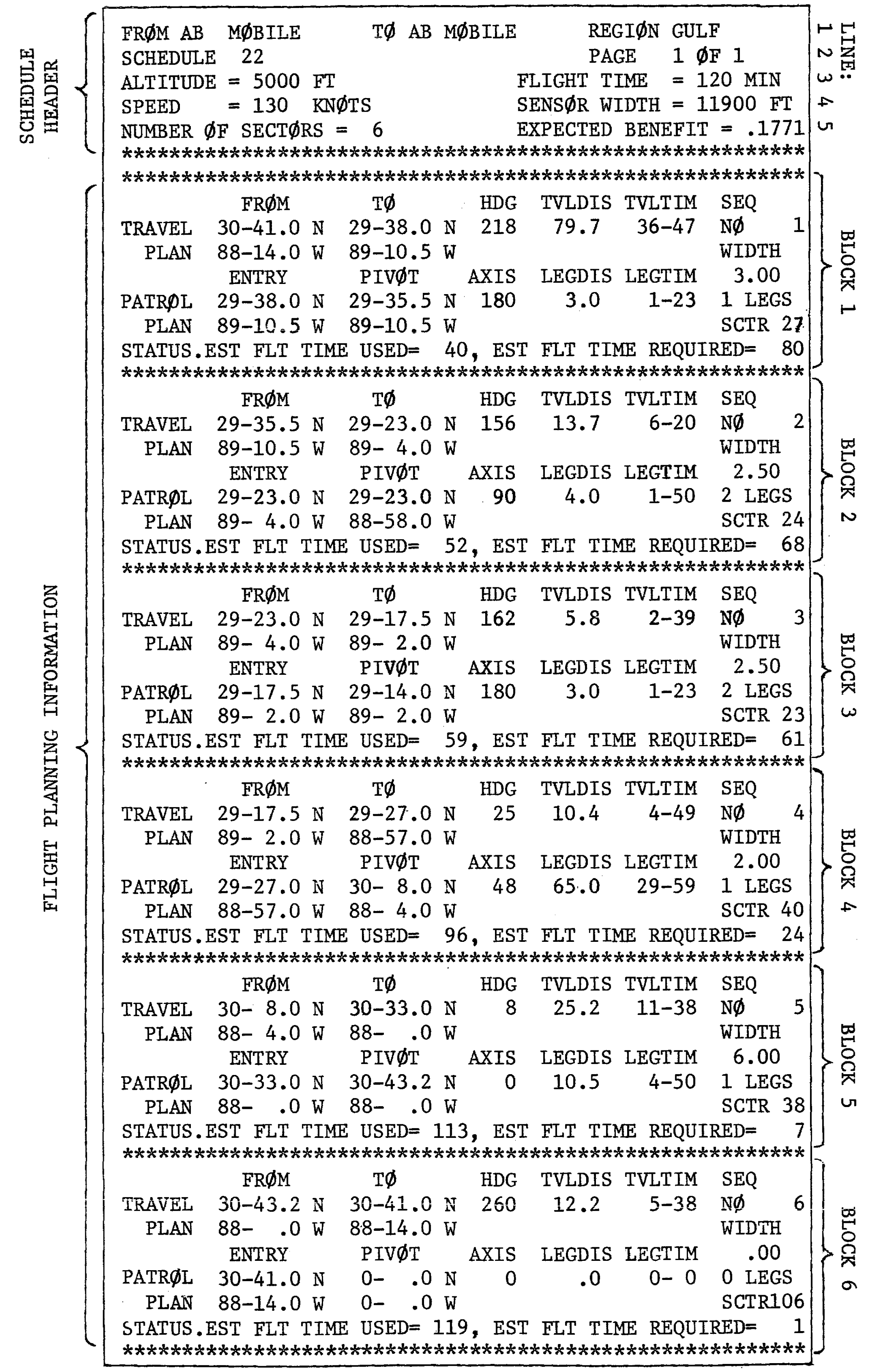
The models and algorithms used in PDAPS Phase II have been implemented in a system of procedures, computer programs, and communicating files which are used to produce actual flight schedules to be used by aircraft in performing pollution surveillance missions. The term OPGENS is an acronym for "Optimal Pollution Surveillance Schedule Generating System." The component parts and interrelationships between the components of OPGENS are shown as a flow chart in Figure 3.

OPGENS can be divided into four subsystems. The first subsystem is identified as Data Collection and Preparation. This subsystem is characterized by considerable manual work and statistical analysis. The essential activities of this subsystem are listed below.

1. Summarize Maritime Administration data for the region.
2. Define sectors for the region.
3. Determine spill data for appropriate sectors.
4. Determine shipment data for appropriate sectors.
5. Specify the sector successor sets.
6. Identify all potential air bases for the region.
7. Define the flight area.
8. Calculate the distance and heading from every sector entry-exit point to the entry-exit points of every other sector in the flight area.

Some data summarization is done by a computer program which reads the shipping data tapes, but otherwise most of these activities are done manually except for the distance-heading calculations which are performed in the OPGENS program, DISTNCE. Notice that activities 1 through 6 in Subsystem 1 are common to all schedules produced for a particular flight region. Thus, the entire Subsystem 1 need be invoked only once for each region and activities 7 and 8 are invoked for each flight area within a region. The manually prepared data is processed, edited, and organized into labelled files for future use by DISTNCE.

The second subsystem consists of three programs, GENERAT, GETDAT, and XFORM, and the files used for communicating between these programs. The primary function is divided into three activities which characterize each of the programs. The activity performed in GENERAT is the generation of the states according to the modified version of Procedure 1B. Another important activity of GENERAT is the defining of all sector-pattern combinations to be used in patrolling the defined sectors. The outputs from GENERAT are passed on to GETDAT where the state transitions are


SCHEDULE HEADER
FLIGHT PLANNING INFORMATION
LINE: 1 2 3 4 5
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 4
BLOCK 5
BLOCK 6

```
FRØM AB  MØBILE        TØ AB MØBILE        REGIØN GULF
SCHEDULE  22                               PAGE   1 ØF 1
ALTITUDE = 5000 FT                  FLIGHT TIME   = 120 MIN
SPEED    = 130  KNØTS               SENSØR WIDTH = 11900 FT
NUMBER ØF SECTØRS =  6              EXPECTED BENEFIT = .1771
*********************************************************
*********************************************************
          FRØM        TØ        HDG   TVLDIS TVLTIM  SEQ
TRAVEL  30-41.0 N  29-38.0 N  218   79.7    36-47  NØ     1
  PLAN  88-14.0 W  89-10.5 W                       WIDTH
         ENTRY       PIVØT     AXIS  LEGDIS LEGTIM   3.00
PATRØL  29-38.0 N  29-35.5 N  180    3.0     1-23  1 LEGS
  PLAN  89-10.5 W  89-10.5 W                       SCTR 27
STATUS.EST FLT TIME USED=  40, EST FLT TIME REQUIRED=  80
*********************************************************
          FRØM        TØ        HDG   TVLDIS TVLTIM  SEQ
TRAVEL  29-35.5 N  29-23.0 N  156   13.7     6-20  NØ     2
  PLAN  89-10.5 W  89- 4.0 W                       WIDTH
         ENTRY       PIVØT     AXIS  LEGDIS LEGTIM   2.50
PATRØL  29-23.0 N  29-23.0 N   90    4.0     1-50  2 LEGS
  PLAN  89- 4.0 W  88-58.0 W                       SCTR 24
STATUS.EST FLT TIME USED=  52, EST FLT TIME REQUIRED=  68
*********************************************************
          FRØM        TØ        HDG   TVLDIS TVLTIM  SEQ
TRAVEL  29-23.0 N  29-17.5 N  162    5.8     2-39  NØ     3
  PLAN  89- 4.0 W  89- 2.0 W                       WIDTH
         ENTRY       PIVØT     AXIS  LEGDIS LEGTIM   2.50
PATRØL  29-17.5 N  29-14.0 N  180    3.0     1-23  2 LEGS
  PLAN  89- 2.0 W  89- 2.0 W                       SCTR 23
STATUS.EST FLT TIME USED=  59, EST FLT TIME REQUIRED=  61
*********************************************************
          FRØM        TØ        HDG   TVLDIS TVLTIM  SEQ
TRAVEL  29-17.5 N  29-27.0 N   25   10.4     4-49  NØ     4
  PLAN  89- 2.0 W  88-57.0 W                       WIDTH
         ENTRY       PIVØT     AXIS  LEGDIS LEGTIM   2.00
PATRØL  29-27.0 N  30- 8.0 N   48   65.0    29-59  1 LEGS
  PLAN  88-57.0 W  88- 4.0 W                       SCTR 40
STATUS.EST FLT TIME USED=  96, EST FLT TIME REQUIRED=  24
*********************************************************
          FRØM        TØ        HDG   TVLDIS TVLTIM  SEQ
TRAVEL  30- 8.0 N  30-33.0 N    8   25.2    11-38  NØ     5
  PLAN  88- 4.0 W  88-  .0 W                       WIDTH
         ENTRY       PIVØT     AXIS  LEGDIS LEGTIM   6.00
PATRØL  30-33.0 N  30-43.2 N    0   10.5     4-50  1 LEGS
  PLAN  88-  .0 W  88-  .0 W                       SCTR 38
STATUS.EST FLT TIME USED= 113, EST FLT TIME REQUIRED=   7
*********************************************************
          FRØM        TØ        HDG   TVLDIS TVLTIM  SEQ
TRAVEL  30-43.2 N  30-41.0 N  260   12.2     5-38  NØ     6
  PLAN  88-  .0 W  88-14.0 W                       WIDTH
         ENTRY       PIVØT     AXIS  LEGDIS LEGTIM    .00
PATRØL  30-41.0 N  0-  .0 N    0     .0     0- 0  0 LEGS
  PLAN  88-14.0 W  0-  .0 W                        SCTR106
STATUS.EST FLT TIME USED= 119, EST FLT TIME REQUIRED=   1
*********************************************************
```

Figure 4—Example of a flight schedule

explicitly defined and organized into a full tree state transition matrix (by rows). This state transition matrix is passed one element (record) at a time to XFORM which transforms the transition matrix to a modified tree structure and maps each state to a unique state number.

The effect of these activities is to reduce the size of the problem and to organize the matrix into a form which can be easily converted to an input matrix for OPTIMA, or which can be conveniently processed by the suboptimization algorithms.

The third subsystem in OPGENS is the schedule generator. Contained within it are three path options. Path 1 proceeds through OPTMTX, OPTIMA, DUMPOP, and SCDGEN where, by setting parameters, either optimization Model III or IV is solved using the *RS* algorithm and schedules are generated. Path 2 subjects the XFORM output to processing through KWIKGN which has the option of producing schedules according to either the *NQ* or *YQ* algorithm. Path 3 uses SKDGEN to produce schedules from XFORM output according to the *LS* algorithm.

The fourth and terminal subsystem in OPGENS consists of one main program, DHEADR, whose principal functions are the calculation and addition of detailed flight (navigation) instructions, and the formatting and printing of the completed schedules. A typical schedule for the Gulf of Mexico is shown in Figure 4. The flight instructions are of two basically different types. Type I consists of instructions for inter-sector travel. Type II provides details to the pilot on how to patrol the sector under surveillance. Each step of the patrol mission is guided by two navigational aids. First, the longitude and latitude coordinates are given for the entry-exit points of each sector visited. Second, the distance, heading, and travel time is given for traveling between sectors. For the patrol portion of a given state transition the sector axis, patrol leg distance, and time to travel one leg are given. All of this information is given for perfect conditions. Thus, the pilot must make adjustments for compass deviations, wind speed and direction, etc.

After all information is generated for a particular surveillance mission the schedule is formatted and printed by the GOULD 4800 electrostatic printer. The size of these schedules is such that they will fit conveniently on the knee boards used by pilots. Considerable research and investigation have gone into the design of the schedule format to insure that the schedules provide the information and instructions needed by pilots for both pre-flight and in-flight activities.

Comparison of algorithms

Three problems of substantially different sizes were generated and solved using the four described algorithms. The first problem was a hypothetical example which had 66 states, 164 state transitions and 1337 non-zero entries in the OPTIMA input matrix. The second problem was a two hour flight out of Mobile, Alabama, and had 341 states, 831 state transitions, and 6724 non-zero matrix entries. The third problem was for a seven hour flight from Mobile to Corpus Christi, Texas, and had 1565 states, 2387 state transitions, and 13252 non-null OPTIMA matrix entries.

The results of the experiment conducted in solving these three problems using each of the four algorithms is presented in Table I.

In the table note that the *RS* algorithm blocks contain two items not found in the others. The row labeled "Execution (w/basis)" contains the OPTIMA execution time when the optimal solution basis was read in as an initial starting basis. This figure is the lower bound time for using a basis generator as previously discussed. The row labeled "(ϵ=.4) Model μ" is interpreted as the value of the objective function from the OPTIMA Model III when .4 is the value for epsilon. The symbol μ is used because this value represents the true mean of the distribution of expected schedule benefits derived from this model. Also, notice that the *LS* algorithm sample size has

TABLE I—Experimental Results

Problem	Sample	2 HR	7 HR
Transitions	164	831	2387
States	66	341	1565
Time	12.3+	141	294.0
RS ALGORITHM			
Non-zero entries	1337	6724	13252
Path time	117	843	8689
Execution time	74	770	8505*
Execution (w/basis)	53	218	690
(ϵ=.4) model μ	2.68	.0885	.2281
Ave. exp. benefit	2.86	.0866	.2223
Variance	.821	.00125	.0035
Std. deviation	.906	.0353	.0588
YQ ALGORITHM			
Path time	4.3	8.65	24.21
Execution time	2.87	7.39	22.89
Ave. exp. benefit	1.240	.0685	.190
Variance	.360	.000847	.0026
Std. deviation	.600	.0291	.0510
NQ ALGORITHM			
Path time	4.2	7.15	22.53
Execution time	3.02	6.12	21.15
Ave. exp. benefit	1.778	.0439	.168
Variance	.590	.000458	.0028
Std. deviation	.768	.0214	.0530
LS ALGORITHM			
Path time	4.01	6.28	11.16
Execution time	3.09	5.12	9.68
Ave. exp. benefit	2.14	.0566	.188
Variance	.675	.000554	.0020
Std. deviation	.822	.0235	.0449
Sample size	50	61	40

* This time is conservative. RS algorithm in solving the 7 hour problem actually took four runs (at one run a day) and about 13,000 seconds. The 8505 seconds reported does not include the set-up time required for the last three of the four runs.

been added in those blocks. The time entry in the header section for each problem is the approximate computation time for running all parts of OPGENS except the scheduling subsystem. This time should be added to the time required by the selected scheduling approach to get the approximate time required to generate a set of schedules according to the given parameters.

The following observations can be made from the results presented in Table I.

(1) The cost of obtaining an optimal solution is significantly greater than any other method. In the sample problem the *RS* algorithm took about 25 times as long as the other algorithms. For the two-hour problem it was longer by a factor of about 100, and for the seven-hour problem it took over 800 times longer than the quickest algorithm.
(2) Schedules resulting from OPTIMA did produce the highest expected benefit. The results indicate that the optimal solution is about 20 percent better than the next best approach.
(3) While the time required for solution by each model is positively correlated with the problem size, the range of solution times required is significantly different for different models. PATH 1 has a very wide range, with the seven-hour problem requiring over 100 times more time than the sample problem. The *YQ* algorithm has a range ratio of about 8 to 1; the *NQ* algorithm about 7 to 1; and the *LS* algorithm about 3 to 1.

In order to compare the efficiency and effectiveness more directly let us define the following two measures.

$$\text{Efficiency }(i) = \frac{\text{Execution time of the fastest algorithm}}{\text{Execution time of algorithm } i}$$

$$\text{Effectiveness }(i) = \frac{\text{Expected benefit of algorithm } i}{\text{Expected benefit of RS algorithm}}$$

Table II presents a comparison of the efficiency factors and is a basis for the following observations.

(1) The *RS* algorithm is extremely inefficient for all problems.
(2) The *LS* algorithm is most efficient for the two real data problems and performs very well on the sample problem.
(3) Both the *YQ* and *NQ* algorithms perform well on the sample problem but decrease in relative efficiency as the problem size increases.

TABLE II—Efficiency Factors

Problem Algorithm	Sample	2 HR	7 HR
RS algorithm	.039	.0067	.0011
YQ algorithm	1.0	.69	.42
NQ algorithm	.95	.84	.46
LS algorithm	.93	1.0	1.0

TABLE III—Effectiveness Factors

Problem Algorithm	Sample	2 HR	7 HR
RS algorithm	1.07	.979	.974
YQ algorithm	.463	.774	.832
NQ algorithm	.663	.496	.737
LS algorithm	.799	.640	.823

Table III shows the effectiveness factors and is a basis for the following comments.

(1) The *RS* algorithm is consistently better than the other methods by at least 20-30 percent.
(2) The *LS* algorithm is consistently more effective than the *NQ* algorithm. (That is, *LS* dominates *NQ* based on efficiency.)
(3) The *YQ* algorithm performs better on the two real data problems than both the *NQ* and *LS* algorithms; however, it is not significantly better than the *LS* algorithm.

From the results presented in Tables I through III it appears that the use of the *RS* algorithm is not a viable approach to solving the surveillance scheduling problem as it has been defined. This conclusion is based on the efficiency of the method. It is apparent that the expected benefit from the *RS* algorithm is not sufficient to justify the substantially greater cost incurred. This appears to be a valid conclusion even if a near perfect basis generator could be devised. The reason for this latter conclusion lies in the fact that the set up and preparation time required by the revised simplex algorithm is too costly, regardless of the actual solution speed. Further support for this conclusion is given by two other experiments not reported in the tables. In one case the seven-hour test problem was solved using a preliminary version of VIVACE' which is a high speed experimental linear programming code being developed at Purdue. VIVACE' took approximately 3800 seconds to achieve optimality. This is still much too costly for production runs. A second supporting case involved the use of a special option in OPTIMA. Within the OPTIMA Application Control Language (ACL) is a verb CRASHX which calls a series of OPTIMA subroutines for selecting a non-trivial starting basis. In this experiment CRASHX was used to obtain an advanced basis to initiate OPTIMA. In this case the solution was optimal after 665 simplex iterations and 1494 central processor seconds. While this is a marked improvement over the standard OPTIMA and VIVACE' it is still too costly in terms of computation time.

Of the three remaining algorithms, any of them will produce suitable, feasible schedules much more efficiently than the *RS* algorithm. The *LS* algorithm appears to outperform the *NQ* algorithm in every measure except for a slight efficiency advantage achieved by the *NQ* algorithm in solving the sample problem.

Beyond this it appears difficult to generalize because the results are mixed, unless some advance knowledge is known about the problem size. Fortunately this is the case. The real problems to be investigated in PDAPS are not likely to be much smaller than the two-hour test problem. If consideration is therefore restricted to the two real data problems then it appears that the *YQ* algorithm is more effective than the *LS* algorithm, but the *LS* algorithm is more efficient than the *YQ* algorithm. While this does not provide a clear decision as to which algorithm to use, it may be appropriate to view this dilemma in perspective. From Table I it can be seen that using any of the three (*NQ*, *YQ*, *LS*) algorithms in the scheduling subsystem will not effect the total OPGENS schedule generation time by more than two percent. Any of these three algorithms represents less than seven percent of the OPGENS time for real data-size problems. One approach may be to run all three algorithms and select the results that are most satisfactory. Otherwise the choice appears to be between the *YQ* and *LS* algorithms.

SUMMARY AND ACKNOWLEDGMENTS

In this paper we have described a system, OPGENS, which has been designed to facilitate the scheduling of fixed wing aircraft used by the U.S. Coast Guard in pollution surveillance activities. The models presented structure the problem as a Markov decision process. The solution algorithms used clearly illustrate the trade-offs between efficiency and effectiveness.

What we have been unable to convey is the magnitude of the data analysis and the human factors engineering effort which has been a part of this effort. We trust that the experienced reader will appreciate that effort.

Finally we wish to express sincere thanks to the U.S. Coast Guard for its cooperation and to LCdr. Ed Demmuzzio and LCdr. Jim Butler for their contribution without which the results would not have been nearly so relevant.

BIBLIOGRAPHY

1. Baumol, William J., *Environmental Protection, International Spill-overs and Trade*, Wicksel Lectures (1971). Stockholm, Almquvist and Wicksell, 1971.
2. Charnes, A. and W. E. Cooper, "Programming with Fractional Functionals: I, Linear Fractional Programming," *Naval Research Logistics Quarterly*, V. 9 N. 3 and 4, pp. 181-186, (1962).
3. Control Data Corporation, *Computer Systems OPTIMA Version 3.0 Reference Manual*, Publication No. 60207000 Rev. B, Minneapolis, Minnesota, (1968).
4. Derman, C. and M. Klein, "Surveillance of Multi-Component Systems: A Stochastic Travelling Salesman Problem," *Naval Research Logistics Quarterly*, V. 13, pp. 103-111, (1966).
5. Dillingham Corporation, Applied Oceanography Division, *Analysis of Oil Spills*, La Jolla, California, 1970.
6. National Search and Rescue Manual, ACC 124/3, FM 20-150, NWP 37(A), AFM 64-2, CG 308, July 1, 1959.
7. United Nations, Food and Agricultural Organization, *Pollution: An International Problem for Fisheries*, Rome, (1971). Especially pp. 15-21.

Use of a micro-computer in a missile simulator

by JAMES A. ROSE and JAMES V. LEONARD

McDonnell-Douglas Astronautics Company-East
St. Louis, Missouri

and

HERBERT A. CROSBY

University of Missouri
Rolla, Missouri

INTRODUCTION

In 1971 McDonnell Douglas Corporation was awarded a contract by the Department of the Navy to develop a new weapon system. The weapon system is comprised of an advanced missile, several Control and Launch Subsystems (CLS) and the support equipment required to test and maintain the missile and CLS.

In a complex advanced missile system the need for some type of missile simulator is obvious. Missile simulators can range in complexity from small, rather simple, manually operated, limited capability devices to major pieces of automatic test equipment capable of simulating the missile and isolating CLS malfunctions to the lowest replaceable component level.

The primary requirement for a missile simulator is to "look like" a missile so that all the Launch System circuitry can be verified quickly and safely. Another desirable characteristic is to provide the capability of simulating a missile "NO-GO" condition to the CLS so that CLS operators can be trained to respond properly to possibly dangerous missile or launcher malfunctions.

MISSILE SYSTEM DESCRIPTION

The types of missile systems which interface directly with the CLS, and therefore must be simulated are, (1) the electrical power system, (2) the discrete command/response system and (3) the digital command/response system. The power system consists of 400 and 60 Hz AC power circuits and several 26 or 28 v-dc circuits. The discrete command/response system consists of the circuits required to process individual 26 v-dc signals which activate missile subsystems and generate the discrete on/off responses which indicate system activation or status. The digital command/response system consists of the digital circuits required to receive and process information in digital form. This information is used to update the missile guidance system and to control all the automatic functions required to launch the missile rapidly and accurately.

The missile digital system interfaces directly with the Missile Data Processor (MDP) in the CLS. The MDP is a digital computer with an 8,000 word memory capability. Its output to the missile consists of three distinct signal types. These are (1) Data Enable, (2) Clock strobe and (3) Digital Data. The timing diagrams for data transmission are shown in Figure 1 and are based on the MDP specification per Reference 2.

Because of the 100 kilobit/second rate of data transmission, a method to verify the accuracy of the received information is required. In this system, odd parity and checksum methods are used to validate all transmitted digital data.

The missile system is capable of producing only two responses to the MDP. These are the "Good Data" word and the "Missile Status" word. If a checksum comparison exists and the parity is valid a "Good Data" response is returned to the MDP. Upon command from the MDP the missile runs through a *built-in-test* (BIT) routine and responds with a "Missile Status" word. Each bit is the GO/NO-GO status of a particular missile system or subsystem.

SIMULATOR DESIGN CRITERIA

Early in 1973 the requirement for an improved missile simulator was recognized and a contract was given to design and build a limited number of development test units. The new simulator was to be called the Missile Simulator/Test Set (S/TS). Prior to starting the detailed design of the S/TS, a set of design criteria were established utilizing the most desirable characteristics of the previous simulator and doing away with the least desirable characteristics. These are:

1. The S/TS shall be portable.
2. The S/TS shall be a self-contained unit requiring no external power source.

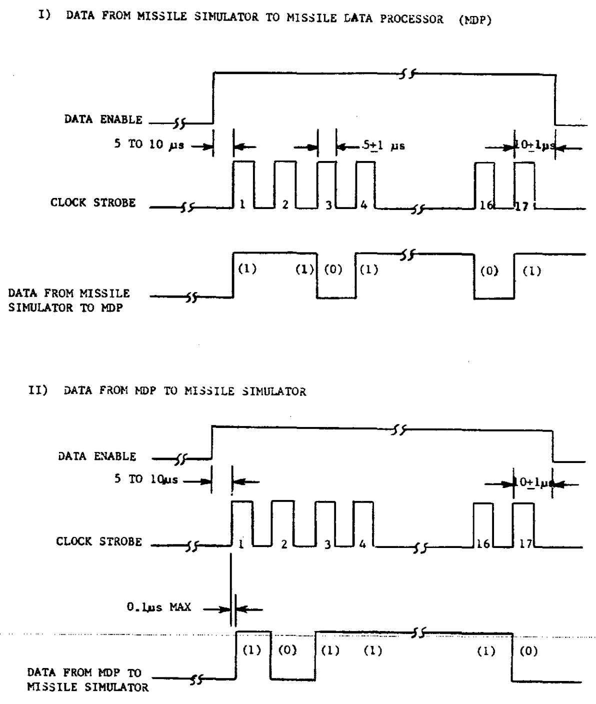

I) DATA FROM MISSILE SIMULATOR TO MISSILE DATA PROCESSOR (MDP)
DATA ENABLE
5 TO 10 μs
5±1 μs
10±1μs
CLOCK STROBE
DATA FROM MISSILE SIMULATOR TO MDP
II) DATA FROM MDP TO MISSILE SIMULATOR
DATA ENABLE
5 TO 10μs
10±1μs
CLOCK STROBE
0.1μs MAX
DATA FROM MDP TO MISSILE SIMULATOR

Figure 1—Missile signal timing

3. The S/TS shall have a load bank circuit which simulates the missile electrical loads on a time shared basis.
4. The S/TS shall be capable of automatic response to selected discrete commands.
5. The S/TS shall be capable of providing a "NO-GO" condition on the discrete output lines by inhibiting the automatic command response at the operator's discretion.
6. The S/TS shall be capable of automatic response to all digital inputs.
7. The S/TS shall have circuitry to validate incoming digital data utilizing the parity and checksum methods.
8. The S/TS shall be capable of simulated "NO-GO" digital responses to the MDP in the following manner:
   (a) by inhibiting the "Good Data" word output at the operator's discretion.
   (b) by simulating a missile subsystem NO-GO condition in the "Status" word at the operator's discretion.
9. The S/TS digital system receivers, drivers, and timing shall be identical to that of the missile digital system.
10. The S/TS shall have a built-in-test (BIT) capability to test the internal loadbank, discrete and digital circuitry.

The S/TS Control Panel is shown in Figure 2.

POWER SYSTEM SIMULATION

In simulating the missile power loads, the S/TS does not use the conventional fixed value, high power dissipation, resistor load approach. It was considered adequate to load the CLS power output circuitry periodically for short spans of time to verify its capability to supply the required current at the stipulated voltage levels. This allows derating the load resistor power ratings as much as three orders of magnitude, depending on required sample duty cycle. This in turn greatly reduces the load bank size and heat dissipation requirements but increases its complexity. It was decided that size and weight were of major concern, hence this approach was adopted for the S/TS.

DISCRETE SYSTEM SIMULATION

In order to alleviate the necessity for a simulator operator to be present for every test, the S/TS was designed to automatically generate the required responses to all discrete commands. This is accomplished by simulating the missile relay loads with electronic relays which are used to control

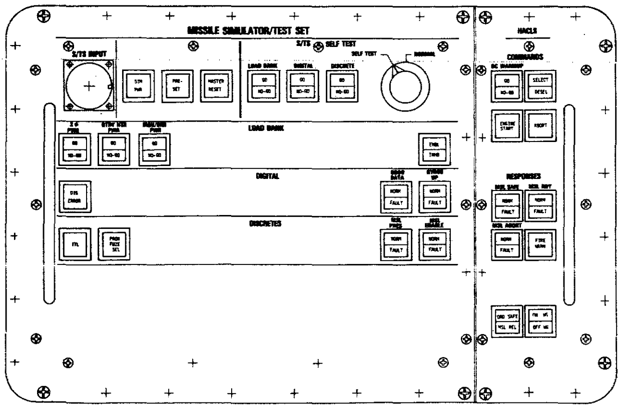


Figure 2—S/TS control panel

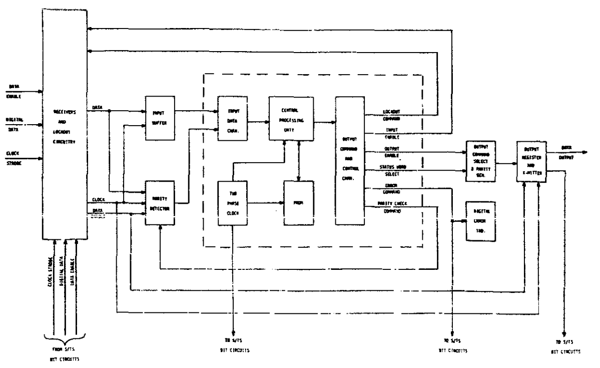


Figure 3—Digital system block diagram

activation of the response signal. Each discrete response can be faulted (i.e., inhibited) by depressing a panel switch indicator. This approach is somewhat more complicated and the hardware more expensive than that of the original simulator but was considered a better simulation of the missile discrete command and response system.

DIGITAL SYSTEM SIMULATION

The shortcomings of the original simulator digital command/response system were eliminated by a complete redesign of the digital section for the S/TS. The basis of the new design is the MCS-4 micro-computer set with additional 256 word, programmable - read - only - memory (PROM). A block diagram of the S/TS digital section is shown in Figure 3.

There is a considerable amount of peripheral logic circuitry required to perform the input/output functions for the micro-computer set. One of the first steps in designing the peripheral logic and generating a workable program for the micro-computer is to prepare a logic tree and from that, a flow chart.

Because of certain format characteristics of the input command words, the logic tree can be made rather simple. (Reference Figure 4.) Each input command word is 16 bits in length and is to be analyzed in bytes of 4 bits each. (Byte 1 is the least significant 4 bits.) The acceptable input command words are one of the following forms:

(a)	$(U000)_{16}$	Request MSL Status
(b)	$(0V00)_{16}$	Reinitiate
(c)	$(00WW)_{16}$	Reset I/O
(d)	$(00XY)_{16}$	Data Group Identifier.

NOTE:

(a) The above command word forms are shown using base 16 format.

(b) For reasons of security, the actual values of "U_{16}", "V_{16}", "W_{16}", "X_{16}", and "Y_{16}" are not given.

LOGIC TREE DESCRIPTION

As can be seen from the above and in Figure 4, if byte 1 is zero, the received word must be of form a) or b) and,

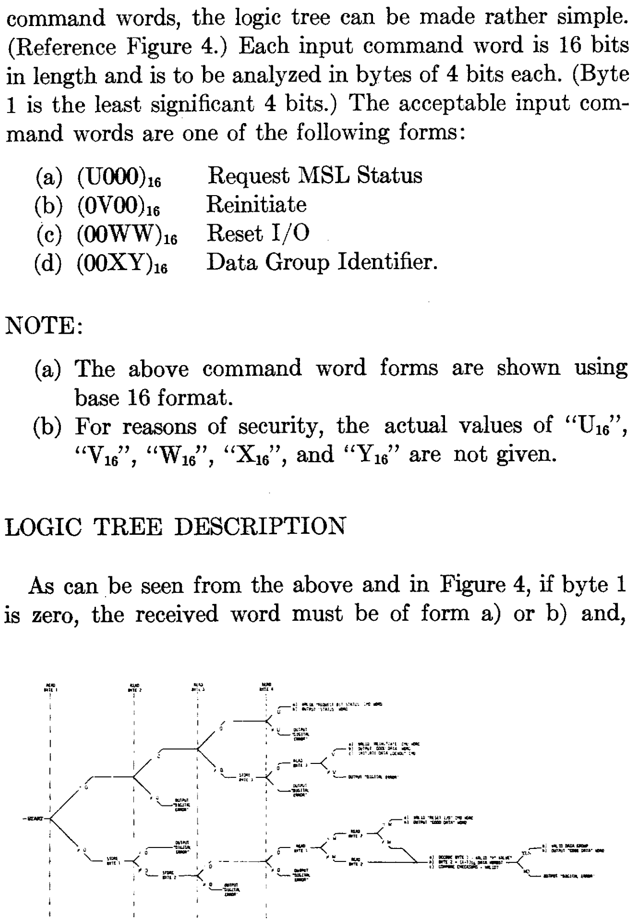


Figure 4—Digital system logic tree

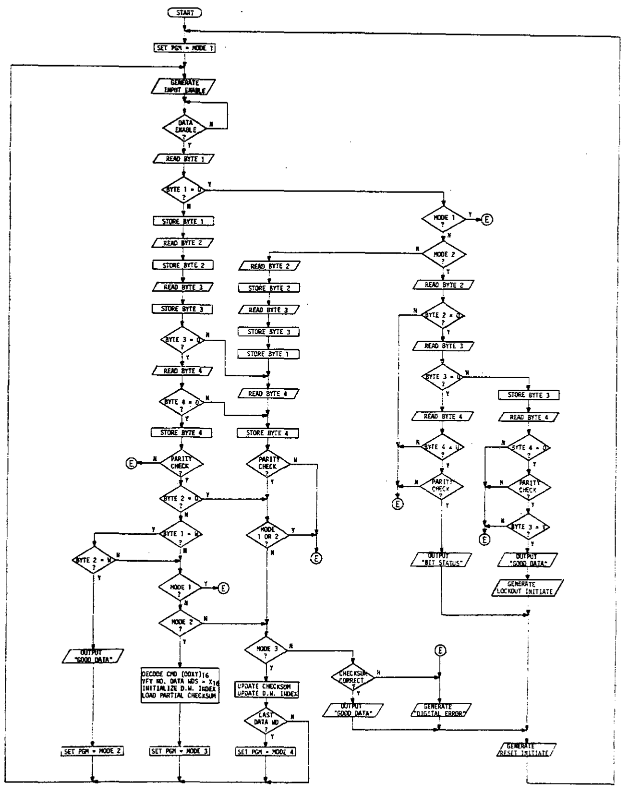


Figure 5—Digital system flow chart

byte 2 must also be zero or an input error exists. If byte 1 and 2 are both zero, only two possibilities exist for bytes 3 and 4; either byte 3 is zero and byte 4 has value "U", indicating a Request BIT Status word has been received or byte 3 has value "V" and byte 4 is zero, indicating the Reinitiate word has been received. All other possibilities are invalid and result in a digital error.

If byte 1 is non-zero, the received word must be of form c) or d), hence byte 2 must also be non-zero. If this is true, then bytes 3 and 4 must be identically equal to zero or a digital input error exists. If bytes 1 and 2 are both of value "W" a Reset I/O word input is indicated. This is because there is no other valid command word which is identically equal in value to $(00WW)_{16}$. If byte 1 and/or byte 2 is a different value than "W", a Data Group Identifier word of form $(00XY)_{16}$ is indicated. The value "Y" for byte 1 is a code indicating a specific block of data is being transferred and the value "X" for byte 2 is the number of words to follow, including the checksum word.

FLOW CHART DESCRIPTION

The following is a description of the S/TS Digital System Flow Chart with explanations as to how some of the functions are to be implemented. As is seen in Figure 5, there are four data processor "modes" mentioned. These are defined as follows:

Mode 1. The data processor program is "set" to expect, and accept, only the "Reset I/O" command word; i.e., $(00WW)_{16}$, which is required to initialize the system. Any other input word will indicate a Digital Error (Note: the "Reset I/O" command is required prior to every input command word).

Mode 2. A "Reset I/O" has been received and the Data processor is "set" to except any command word. (Including "Reset I/O")

Mode 3. A valid command word has been received. If it is of form $(00XY)_{16}$, set the data processor to expect X_{16}—1 data words.

Mode 4. The last data word (X_{16}—1) has been received, the data processor is set to except the checksum word as the next transmission.

In Figure 5 it can be seen that initially the processor is set to mode 1. A Data Input Enable command is generated in the Processor Output Command Channel. The system remains in this state waiting for a "Data Enable" from the MDP. With receipt of a "Data Enable," the accompanying input data is clocked serially, least significant bit (LSB) first, into the Input Buffer (I/O register). As the first four bits enter the register, they are read out in parallel "on-the-fly". If byte 1 does not equal zero, it is stored and bytes 2 and 3 are read and stored. (Reading the data in bytes as it is being clocked in instead of storing each byte and then going back to recall it, read it, and re-store it saves considerable processing time and I/O hardware.)

The fact that byte 1 does not equal zero indicates that the incoming word is a "Reset I/O" command word, ($00WW_{16}$), a "Data Group Identifier" word, $(00XY)_{10}$, or some invalid command word. The "Reset I/O" and "Identifier" words both require that bytes 3 and 4 be equal to zero. If byte 3 is zero, byte 4 is read and stored. If byte 4 is also zero a parity check is made of the word. If the parity is even a Digital Error exists. If it is odd, byte 2 is recalled and read. If byte 2 is non-zero, bytes 1 and 2 are then checked to see if they each equal "W". If so, a Good Data" word response is generated and clocked out to the MDP. The S/TS processor program is then set to mode 2. If bytes 1 and/or 2 are not equal to "W", the processor mode would be checked. If the processor is in mode 1; i.e., expecting a "Reset I/O" command, a digital error is indicated. If the word preceding the incoming command word had been a valid "Reset I/O" command, the processor would now be in mode 2, hence the incoming word must be a "Data Group Identifier" word.

The "Identifier" word "Y" value is evaluated and compared to a listing of acceptable Y values. The "X" value is evaluated and a Data Word Index Counter is initialized to the value (X_{16}—1). A partial checksum consisting of the "value" of the Identifier word is jam transferred from the I/O register into a parallel adder. (Note: this type of adder is such that each sum is retained and the next addend is added to it, keeping a running total). After the Data Word Counter is initialized and the partial checksum loaded the processor program is set to mode 3.

Assume that the next input word received has byte 1 equal to zero. Since the processor is not in mode 1 or 2, the received word is assumed to be a data word and all four bytes are read "on-the-fly" and stored in the I/O register. Odd parity of the input word is verified and the processor mode is rechecked. If even parity exists or the processor had been in mode 1 or 2, a digital error is indicated. With the processor in mode 3, the Data Word Counter is decremented by one, and the data word "value" is jam transferred into the checksum adder. If the Data Word Counter has not been decremented to zero, the last data word has not been received and the processor remains in mode 3.

The remaining data words in the Data Group can be of any "value" hence, bytes 1, 2, 3, and 4 may or may not equal zero. If byte 1 does not equal zero the same path in the Flow Chart as was followed for the "Reset I/O" or "Identifier" word inputs is again followed. The difference is that if bytes 3 or 4 do not equal zero, or byte 2 does equal zero, the logic flow is laterally transferred to a like functional location on the path followed by the first Data Word.

When the Data Word Counter is decremented to zero, the last data word has been received and the processor program is set to mode 4. The next word received, therefore, should be the checksum word. The checksum can, obviously have any "value", hence can follow any one of the logic flow paths described previously, eventually getting to the mode 3 check in the lower center of the flow chart. Since the processor is no longer in mode 3, a checksum comparison is made. To make the checksum comparison the received checksum is clocked into the I/O Register, the register is complemented and the checksum complement is jammed into the adder. If the adder sum is all "1s" a valid checksum is indicated and a "Good Data" response is generated and clocked out to the MDP.

Assume byte 1 equaled zero and the processor was in mode 2. It is obvious that only two valid command words exist under these conditions. These are the "Reinitiate" command, $(0V00)_{16}$, and the "Request Bit Status" word, $(U000)_{16}$. For both of these words, byte 2 must also equal zero or a digital error is indicated. If byte 3 equals zero, byte 4 must equal "U" or a digital error exists. Parity is checked on the received word and if it is valid the "BIT Status" word is generated and clocked out to the MDP.

If byte 3 does not equal zero, then byte 4 must equal zero or a digital error is indicated. After the parity check, byte 3 is compared to the value "V". If they are equal a "Good Data" word is generated, a 50 msec Input Data Lockout is initiated and the processor is reset to mode 1 to await a "Reset I/O" command.

The digital system design and micro-computer program dictated by the Logic Flow has been simulated on the McDonnell-Douglas Automation Company SIGMA 47 computer. The chart just discussed is the result of two previous flow chart simulations which pointed out several timing and sequencing problems. Use of the micro-computer and the results of the simulations make the S/TS Engineers confident the unit has been designed to make maximum use of available hardware and will accurately simulate the missile.

SUMMARY

In 1971 McDonnell-Douglas was awarded a contract for a new missile system. The missile power, discrete command/response, and digital systems were described with emphasis on the digital data format, error detection and timing. A missile simulator, the S/TS, was briefly described showing how the shortcomings of a previous simulator were eliminated. A listing of the S/TS design criteria was given and the design logic based on utilization of a micro-computer was described.

ACKNOWLEDGMENTS

The authors wish to express their appreciation to Mr. Bruce V. Savage and William G. Kull, who were given the detail design responsibility for the S/TS circuitry and micro-programming, for their advice in preparing the figures used in this paper. Also the authors wish to recognize Mr. James E. Moore of McDonnell Douglas and Mr. Lowell Noxin of the Naval Air Systems Command for their help in establishing the basic design criteria for the S/TS.

REFERENCES

1. Robinson, C. A., Jr., "Harpoon Slated for Prime Anti-ship Roll," *Aviation Week and Space Technology,* May 7, 1973.
2. Kropiunik, F. J., *Harpoon Data Processor, Specification for:,* McDonnell Douglas, September 1971.
3. Rose, J. A., *Harpoon Missile Simulator/Test Set, Critical Design Specification for:,* McDonnell Douglas, January 1974.

Charge-coupled devices for computer memories

by J. E. CARNES and W. F. KOSONOCKY

RCA Laboratories
Princeton, New Jersey

and

J. M. CHAMBERS and D. J. SAUER

RCA Laboratories
Van Nuys, California

INTRODUCTION

The charge-coupled device (CCD)[1] is an analog shift register which is based upon MOS (metal-oxide semiconductor) technology and is capable of large scale integration. As an analog device it offers exciting and unique capabilities as an image sensor and in a variety of signal processing and filtering applications.

The CCD can also be utilized as a digital serial memory where it offers potential cost, size, and power advantages. All of these CCD advantages arise from the fact that only two or three MOS capacitors are required to store one bit of information. Area packing densities of 10^6 bits per square inch seem to be possible along with single chips having 32,000 or more bits of storage.

This paper is intended to review the basic operation of the CCD, the types of memory systems which have been considered to date, and the expected performance of CCD memories including the fundamental design trade-offs possible.

THE BASIC CCD SHIFT REGISTER

MOS capacitor

Since a CCD is physically just a linear array of closely-spaced MOS capacitors, it is important to understand the MOS capacitor and how the surface potential, V_s (the potential at the Si-SiO$_2$ interface relative to the potential in the bulk of the silicon), depends upon the various parameters involved.

Figure 1 shows a cross-sectional view of an MOS capacitor with a p-type silicon substrate. When a positive step voltage is applied to the gate of such a structure, the majority carriers, holes, are repelled and respond within the dielectric relaxation time. This results in a depletion region of negatively-charged acceptor states near the surface of the silicon. Just after the step voltage is applied to the gate, the silicon conduction band at the surface is well below the equilibrium Fermi level and electrons, the minority carriers, will tend to gather there. However, it takes a rather long period of time for thermally-generated minority carriers to accumulate in sufficient numbers to return the system to thermal equilibrium. Thermal relaxation times for MOS capacitors ranging from 1 to 100 seconds have been measured. When minority carriers do accumulate at the surface, they start to create an inversion layer which resides within 100Å of the interface. This negative charge tends to reduce the surface potential, V_s. When V_s goes to zero, no more charge can be accumulated or stored in the potential well. The capacitance of the potential well, C_{well}, consists of two capacitances in parallel: the fixed oxide capacitance and the capacitance associated with the depletion layer of the silicon. Thus, the following fluid model of the MOS capacitor emerges: a potential well for minority carriers can be created by applying a step voltage to the gate and this well will take a relatively long period of time to accumulate charge thermally. For times much shorter than this thermal relaxation time, a potential well exists at the surface, and the depth of this well can be altered by changing the gate voltage. When minority carriers are introduced as signal charge in the potential well, they tend to reduce the depth of the well according to Q_{sig}/C_{well} so they tend to fill up the well much like fluid in a container.

Basic charge-transfer action

A three-phase CCD is just a line of these MOS capacitors spaced very close together with every third one connected to the same gate, or clock voltage as shown in Figure 2(a). If a higher positive voltage is applied to the ϕ_1 clock line than ϕ_2 and ϕ_3, the surface potential variation along the interface will be similar to Figure 2(b). If the device is illuminated by light, charge will accumulate in these wells. Charge can also be introduced electrically at one end of the line of capacitors from a source diffusion controlled by an

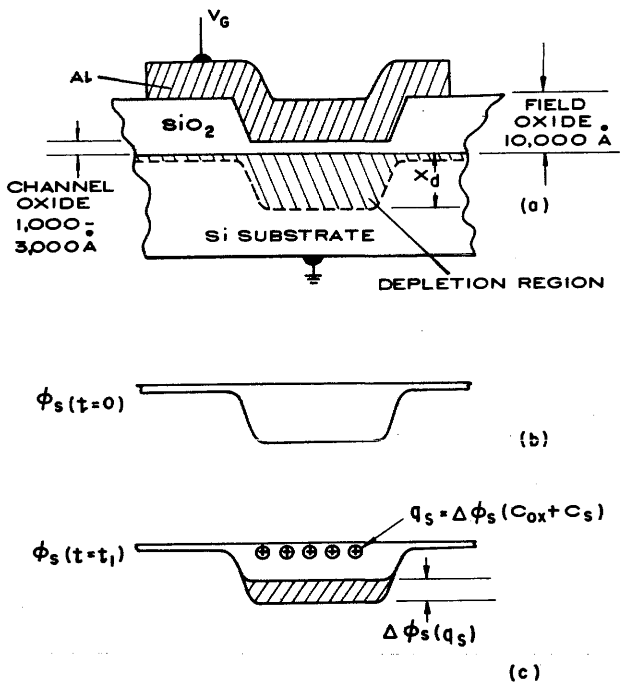


Figure 1—(a) Cross-sectional view of an MOS capacitor representing element for charge-coupled circuit; (b) Surface potential profile just after application of step voltage V_G. (c) Surface potential profile with charge signal q_s in the potential well

input gate. To transfer this charge to the right to the position under the ϕ_2 electrodes, a positive voltage is applied to the ϕ_2 line. The potential well there initially goes deeper than that under a ϕ_1 electrode, which is storing charge, and the charge tends to move over under the ϕ_2 electrodes. Clearly, the capacitors have to be close enough so that the depletion layers overlap strongly, and the surface potential in the gap region is a smooth transition from the one region to the other. Next, the positive voltage on the ϕ_1 line is removed to a small positive DC level, enough to maintain a small depletion region, increasing the surface potential under the ϕ_1 gates in the process. Now the ϕ_2 wells are deeper, and any charge remaining under ϕ_1 gates spills into the ϕ_2 wells. The charge, at least most of it, now resides one-third of a stage to the right under ϕ_2 gates. The charge is prevented from moving to the left by the barrier under the ϕ_3 gates. A similar process moves it from ϕ_2 to ϕ_3 and then from ϕ_3 to ϕ_1. After one complete cycle of a given clock voltage, the charge pattern moves one stage (three gates) to the right. No significant amount of thermal charge accumulates in a particular well because it is continually being swept out by the charge transfer action. The charge being transferred is eventually transferred into a reversed-biased drain diffusion and from there it is returned to the substrate. The charging current required once each cycle to maintain the drain diffusion as a fixed potential can be measured to determine the signal magnitude (current-sensing) or a re-settable floating diffusion which controls the potential of a MOSFET gate can be employed (voltage-sensing).[2] One can visualize a CCD shift register as a multi-gate MOSFET in which the charge signal is moved as charge packets from the source diffusion to the drain diffusion under the control of phase clock voltages applied to the gates.

Limitations on speed and efficiency

Clearly, 100 percent of the charge cannot move instantaneously from one potential well to another. Also some of the charge gets trapped in fast interface states at each site and cannot move at all. Therefore, in a given clock period not quite all of the charge is transferred from one well to the next. The fraction of the total that is transferred (per gate) is called the transfer efficiency, η. The fraction left behind is the loss per transfer, or transfer inefficiency, denoted ϵ, so that $\eta + \epsilon = 1$. Because η determines how many transfers can be made before the signal is seriously distorted and delayed, it is a very important figure of merit for a CCD. If a single charge pulse within initial amplitude P_o is transferred down a CCD register, after n transfers the amplitude P_n will be:

$$P_n = P_o\eta^n \cong P_o(1 - n\epsilon) \qquad \text{(for small } \epsilon) \qquad (1)$$

Clearly, ϵ must be very small if a large number of transfers are required. If we allow an $n\epsilon$ product of 0.1, an overall loss of 10 percent, then a 3ϕ, 330 stage shift register requires $\epsilon \leq 10^{-4}$, or a transfer efficiency of 99.99 percent.

The maximum achievable value for η is limited by how fast the free charge can transfer between adjacent gates[8] and how much of the charge gets trapped at each gate location by fast interface states.[9]

Several physical mechanisms cause charge motion from one potential well to an adjacent deeper well, including

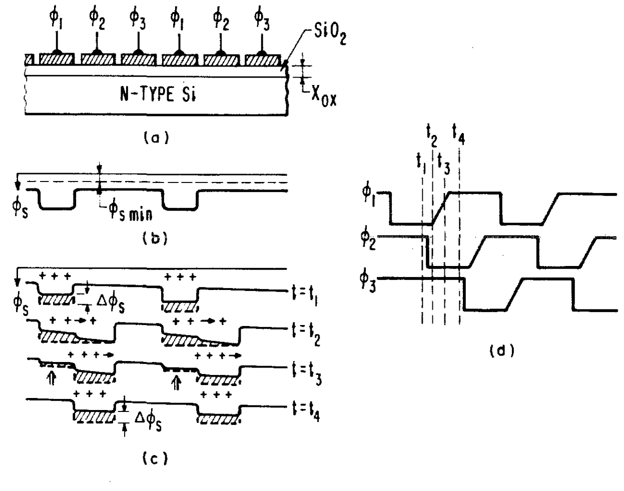


Figure 2—Operation of a 3-phase charge coupled shift register: (a) Cross section of the structure along the channel oxide; (b) Surface potential profile for $\phi_1 = -V$, $\phi_2 = 0$, $\phi_3 = 0$ forming a potential well under the phase-1 electrode; (c) Transfer of charge from the potential wells under the phase-1 electrode to the potential wells under the phase-2 electrode illustrated by the profiles of surface potential at times shown in (d); (d) Waveforms of the phase voltages

charge repulsion,[3] thermal diffusion,[4] and drift under the influence of fringing fields induced by the gate voltages.[5] The rate of transfer depends mainly upon the gate length along the direction of charge movement. It also depends to a lesser degree upon substrate doping, clock waveforms and other structural parameters. Computer simulations indicate that conventional surface channel devices with 10 μm gate lengths are capable of efficient operation at 5-10 MHz.[6] Experimental devices with 7.5 μm gate lengths have been operated at 20 MHz with transfer inefficiencies of 10^{-3}.

Charges can be lost from the signal into fast interface states because, while the filling rate of these states is proportional to the number of free carriers, emptying rate depends only upon the energy level of the interface state. Thus, even though a roughly equal amount of time is available for filling as for emptying, many of the interface states can fill much faster than they can empty, and thus retain some of the signal charge and release it into trailing signal packets.[7] This type of loss can be minimized by continually propagating a small zero-level charge or "fat zero" through the device. This tends to keep the surface states filled so they do not have to be filled by the signal charge.

Buried channel CCD's

The transfer loss due to interface state trapping and the subsequent requirement for fat zero charge can be eliminated if the potential well could be moved away from the Si-SiO$_2$ interface. This is done in the buried channel CCD by including a thin layer of conductivity type opposite to that of the substrate as shown in Figure 3(a). When this layer is completely depleted of majority carriers by applying the appropriate potential to the drain diffusions, the depleted layer results in a parabolic potential well as seen in Figure 3(c). This well can store and transfer charge as described earlier.

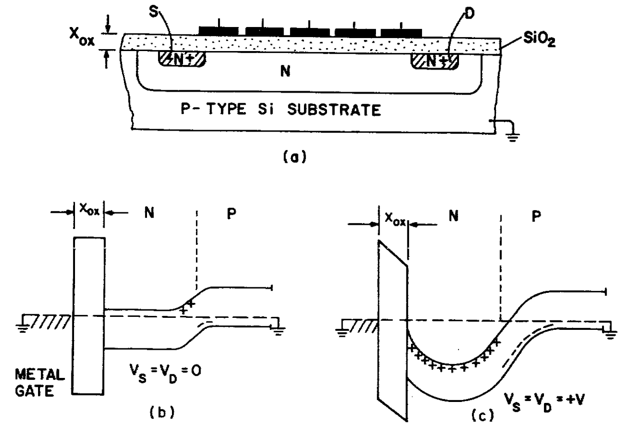


Figure 3—Buried-Channel CCD: (a) Crossection of the device; (b) Energy-band diagram in thermal equilibrium; (c) Energy-band diagram where the n-type layer is depleted

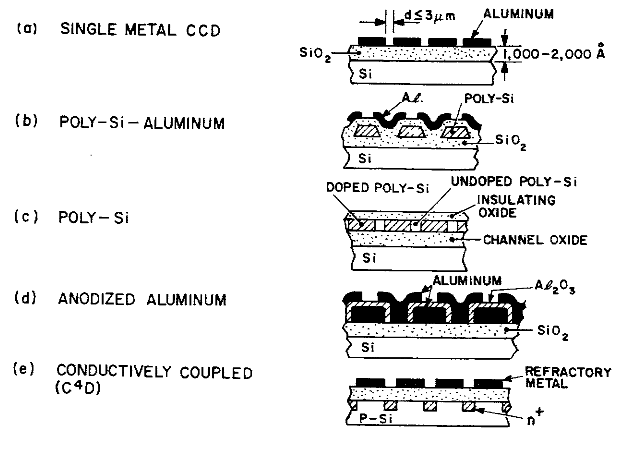


Figure 4—Charge-coupling structures

standard silicon gate processing. The main limitation of this approach is the RC time delay in long gates due to the high resistivity of the polysilicon.[12]

Since the signal charge is then no longer subjected to fast interface state trapping, there is no requirement for fat zero. In addition the carrier mobility is higher since the charge transport occurs in the bulk rather than at the surface, and fringing fields are higher since the electrodes are further away. Thus buried channel devices should be faster,[9] not require fat-zero, and not suffer from edge effect trapping.

On the other hand, buried channel devices involve somewhat more complex processing and more critical operation. They also have reduced signal handling capability because the well capacitance is smaller. Bulk trapping states may also affect buried channel operation at certain resonant frequencies.

CCD technologies

In order to fabricate a CCD one must decide upon:

(1) type of charge-coupling gate structure,
(2) means for channel confinement,
(3) surface vs. buried channel, and
(4) substrate resistivity.

Figure 4 shows five different types of gate structures. Figure 4(a) shows a single level metal (three-phase) structure where the gaps between electrodes are determined by the etching process. While this structure was used to make early CCD's,[10] the instabilities caused by the exposed gap make its use in future devices, especially memories, unlikely.

Figure 4(b) shows a two-phase polysilicon aluminum gate structure which has self-aligned gaps and no exposed oxide to cause instability.[11–12] Standard layout rules (.2 mil etch, .1 alignment) results in 1.2 mil center-to-center spacing between CCD stages. This structure employs essentially

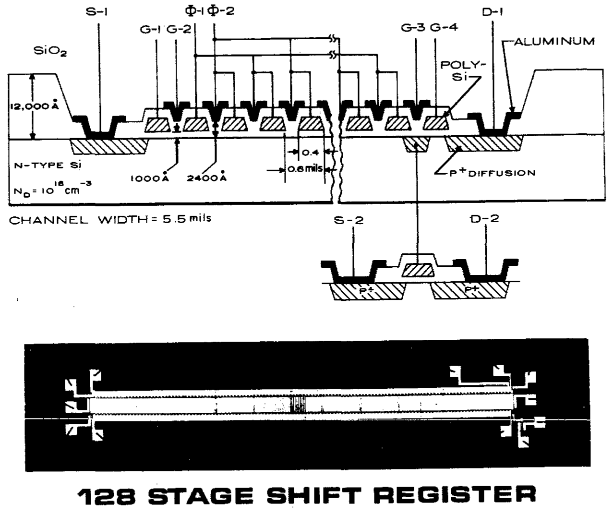


Figure 5—Cross-sectional view and labeled photograph of 128-stage 2-phase shift register

A means of avoiding the exposed gap problem associated with the single level metal approach is shown in Figure 4(c). A layer of polysilicon is doped in the gate regions and is left undoped in the gap regions.[13]

Another multi-level structure is shown in Figure 4(d). Here both gates are aluminum with the insulator being a wet-anodized aluminum oxide.[14] This approach has the advantage of being free of clock phase delay and attenuation. Finally, a fifth approach which is potentially useful for memories is the conductively-connected charge-coupled device C4D (Figure 4(d)) which obviates the need for narrow gaps by ion implanting a conducting layer in the

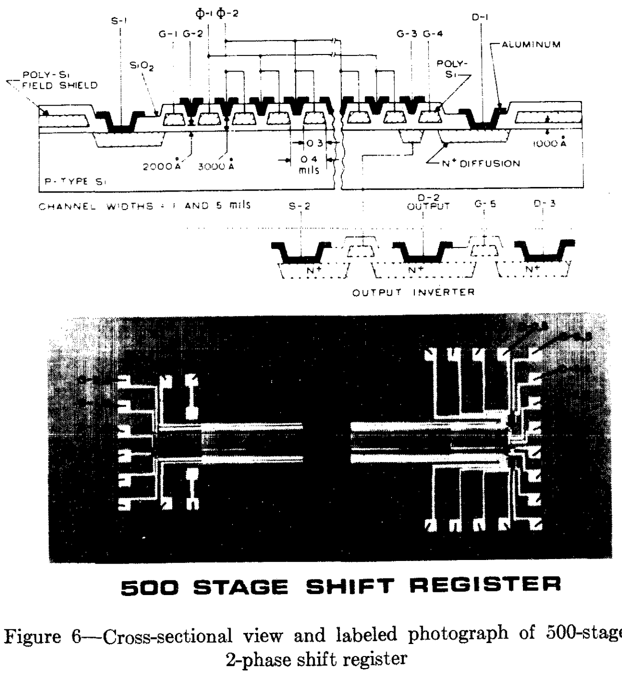


Figure 6—Cross-sectional view and labeled photograph of 500-stage 2-phase shift register

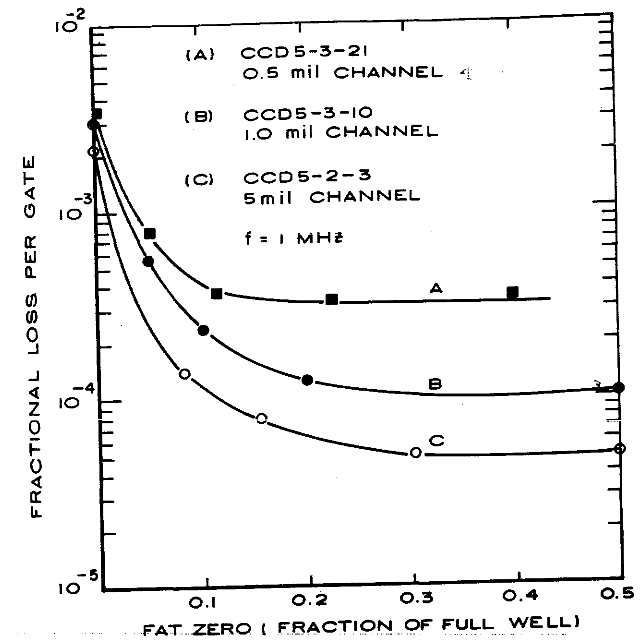


Figure 7—Fractional loss per transfer (transfer inefficiency) versus amount of fat zero for 0.5, 1.0, and 5.0 mil wide 128-stage 2-phase registers. These devices were made on 1.0 ohm-cm n-type silicon substrates with (100) orientation

gap region.[15] Thus a single level metal with standard .2 mil gaps can be used.

The various approaches possible for channel confinement include thick oxide, diffused channel stop, and polysilicon field shield. The thick oxide approach is most useful for p-channel devices, but can cause problems with fine line definition in subsequent processing steps since mask contact cannot be made with the wafer surface. Diffused channel stops avoid this problem, but require more area in output circuits since a minimum separation between source-drain diffusions and channel stops must be maintained. Polysilicon field shield permits close spacing in output circuits and can be used with both p- and n-channel devices. Regardless of the approach used, channel stop widths of .2 mil are sufficient to prevent spillage of charge between adjacent channels.[12]

The choice between surface channel and buried channel for memories is not clear at present. Surface channel offers high charge capacity per unit area but requires a fat zero for efficient operation. The buried channel device basically does not need fat zero but has lower capacity.

The choice of substrate resistivity for surface channel devices involves a trade-off between higher speed and larger transfer loss for high resistivity substrates vs. lower speed and lower transfer loss for low resistivity. This trade-off is illustrated further in the experimental results section. The choice of substrate resistivity is also important in achieving proper operation of two-phase devices using two-thicknesses of oxide to achieve directionality.[16]

The appropriate choice of the various options depends

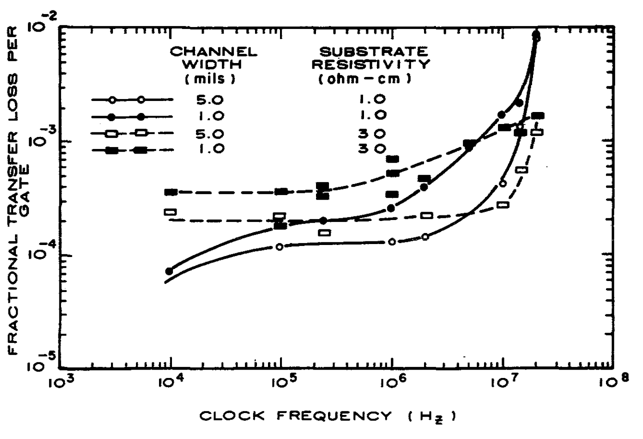


Figure 8—Fractional transfer loss per gate (transfer inefficiency) for 500-stage 2-phase registers operated with 30 percent of fat zero

to a great extent upon the fabrication capabilities one has at his disposal. For those possessing standard silicon gate processing capability, the polysilicon-aluminum gate two-phase device appears to be a particularly advantageous approach—especially for memory structures where phase delay problems associated with rather high sheet resistance of polysilicon are not significant.

EXPERIMENTAL RESULTS

A wide variety of CCD's have been built and results reported.[9,10,12,14,17–19] In this section, typical surface channel device performance will be illustrated using the two-phase polysilicon-aluminum gate device[12] as an example. A number of different configurations have been built including 128- and 500-stage registers, p- and n-channel, various substrate resistivities and channel widths. Figure 5 shows a cross-sectional view of the 128-stage device which has 1.2 mil center-to-center spacing while Figure 6 shows a similar view of the 500-stage device which has .8 mil spacing.

Figure 7 shows the importance of fat zero for efficient operation of surface channel devices. Also apparent in this figure is the edge effect,[20] illustrated by the fact that the narrower channels do not achieve as low a transfer loss as the wider channel devices. Figure 8 illustrates the transfer inefficiency as a function of clock frequency for the 500 stage device with fat zero. The higher resistivity substrates provide higher speed but the lower resistivity substrates achieve lower transfer loss at lower frequencies. This experimental data shows that transfer inefficiencies of 10^{-4} can be achieved with surface channel devices with fat zero charge present. Buried channel devices are, in principle, capable of such operation without fat zero. More performance data on buried channel operation will be available in the near future.

CCD MEMORY SYSTEMS

Charge-coupled shift registers are basically analog devices with no signal gain mechanism. To use these devices for the storage of digital signals, it is necessary to periodically refresh or to regenerate the charge signal. In principle, simple charge regeneration stages, such as shown in Figure 9, are possible and have been experimentally demonstrated.[2,11] Floating diffusion or floating gate sensing can be used. A simple direct-coupled stage will not, however, be sufficient in actual operating memory devices. More complicated regeneration stages are necessary to provide for the introduction of fat zero, to provide for gain and to shift d.c. voltage levels. Experiments using stages similar to that shown in Figure 9 indicate that the operation of such a stage is predictable on the basis of the various capacitances shown. Design of more involved stages can therefore follow conventional MOS circuit design procedures. The importance of gain in the regeneration stage increases as the unit cell area and its associated capacitance becomes small compared to the interconnect capacitances associated with the regeneration stage. Care must be taken not to degrade the inherently high signal-to-noise ratio of the CCD shift register in the regeneration stage.

Various system organizations have been considered for the construction of charge-coupled memories. The choice of the system organization depends on the desired system performance. Two types of systems which can be used to form the basic memory loop are illustrated in Figure 10. As is shown, the signal flow in system A follows a serpentine pattern and has signal refreshing stages at each corner. In this serpentine system, all bits traverse the same path through the loop at the same frequency. Each bit passes through every storage site in one trip around the loop. The number of bits between refresh stages is determined either by transfer efficiency or the lowest operating frequency desired in the standby or idle mode of operation. The num-

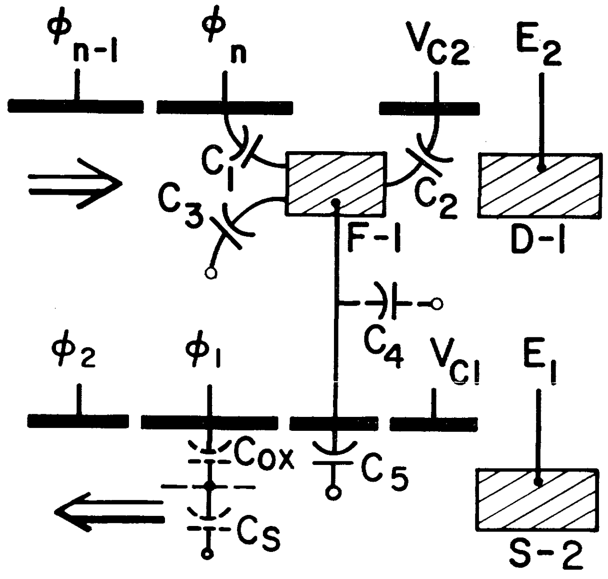


Figure 9—Schematic diagram of a charge regeneration (refreshing) stage illustrating the significant capacitances associating with the floating diffusion used for voltage sensing of the output

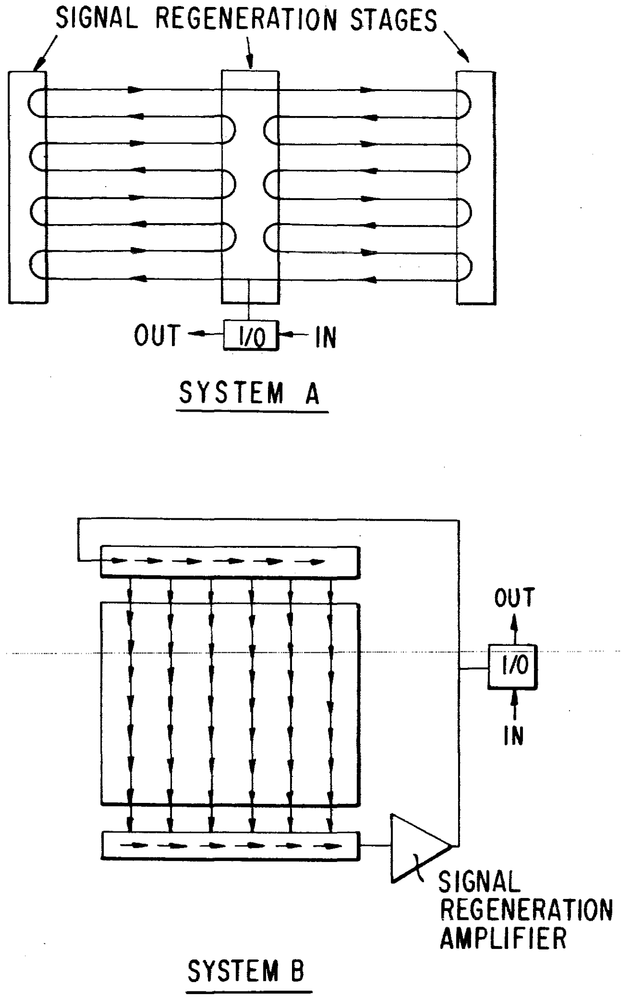


Figure 10—CCD Memory systems: (a) Serpentine; (b) Series-Parallel-Series (SPS)

ber of stages between I/O stages determines the average access time.

Figure 11 shows a possible layout using a multi-level gate technology for such a serpentine structure. The direction of charge motion alternates from one horizontal channel to the next.

System B in Figure 10, on the other hand, is in the form of two serial and one large multi-channel parallel shift registers. The data string is introduced serially into the top serial high speed register. Once loaded, it is transferred in parallel into the middle register. All of the vertical parallel channels in the middle area are clocked together at a lower frequency. At the output the process is reversed, the lower high speed serial register is loaded in parallel and read-out serially to a regeneration and I/O stage. In this series-parallel-series (SPS) system, all of the bits do not traverse the same path. If there are N_x bits across and N_y bits down, then each bit is transferred through $N_x + N_y$ stages and N_y of them are at a lower clock frequency. Thus a single regeneration stage can support a larger number of bits than is possible in the serpentine system.

The SPS system can also be built with a vertical channel for each phase of the horizontal serial registers with every other vertical channel being loaded during alternate read-ins of the serial register. For a two-phase system, the first horizontal line would be stopped under phase-one electrodes and transferred into every other vertical channel. The second line would be stopped on phase-two electrodes and transferred into the remaining vertical channels. The full line would then be transferred down one stage and the procedure repeated. This "interlace" mode would result in twice the packing density possible with a standard SPS structure with one vertical channel per horizontal stage.

Collins *et al.* have proposed another CCD-like structure for increasing the effective packing density of charge transfer memories.[21] Using a conventional CCD shift register, one bit of storage in a two phase system requires two storage wells for each bit. Thus, one-half of the potential storage wells are unoccupied at all times. All storage sites but one could be occupied if each site had its own clock and, starting at the output, each bit was transferred sequentially into the empty slot. The empty slot would appear to move from the output toward the input. This is called the electrode per bit (E/B) scheme. A similar structure called the multiplexed electrode per bit CCD (ME/B) reduces the required number of clocks and increases the data rate possible with the E/B approach. No known devices of this type have been fabricated.

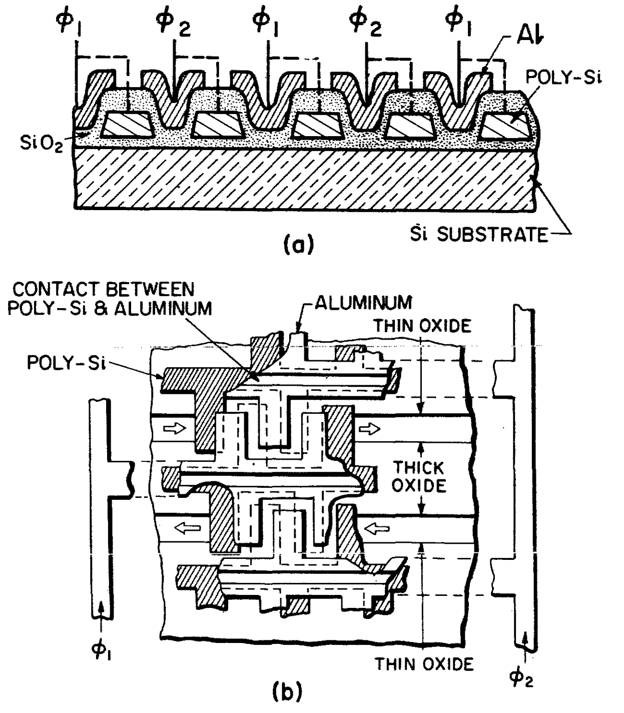


Figure 11—One possible layout for 2-phase, serpentine, CCD memory

CCD's have also been considered for use with MNOS structures as static, non-volatile memories.[22] The presently reported data suggest that the combination of the CCD and NMOS technology for the construction of such read-mostly memories very well may turn-out to be a marriage of inconvenience.

EXPERIMENTAL CCD MEMORY ARRAY

An experimental CCD memory chip employing the SPS configuration is shown in Figure 12. It is constructed using the two-phase aluminum-polysilicon gate structure.[12] The chip is comprised of five independent 1024 bit SPS segments, each having 16 bit serial input and output registers, 64 bit parallel registers, and a regenerate amplifier. Several additional test features are also included on the chip. Figure 13 shows the detected output waveforms from two of the 1024 bit cells connected serially by means of a signal regeneration stage. The operating frequency is 1 MHz and fat zero operation is used.

CCD MEMORY PERFORMANCE

Cost, packing density, and chip size

The main interest in CCD memories is their potential for low cost per bit. These lower costs are expected to be achieved by putting more bits on a chip while maintaining reasonable manufacturing yields. The CCD memory should result in more bits per chip due to two separate factors. First is the smaller area required per bit. Second is the simpler processing which does not require a large number of contact openings or diffusions. The simpler processing should permit acceptable yield on larger chip sizes than is possible with conventional MOS LSI.

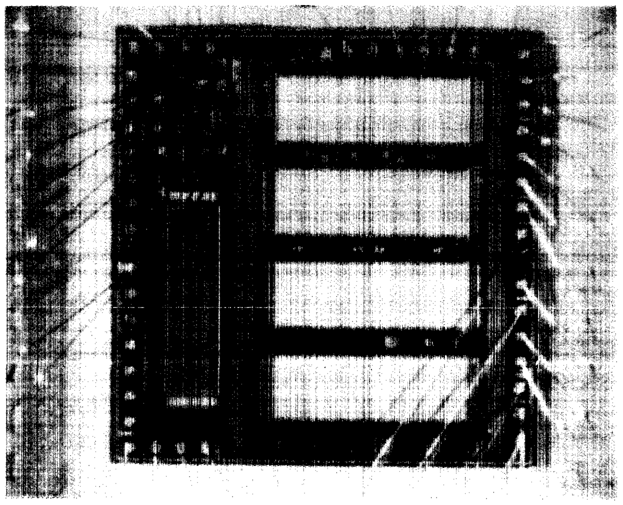


Figure 12—Photomicrograph of an experimental CCD memory array

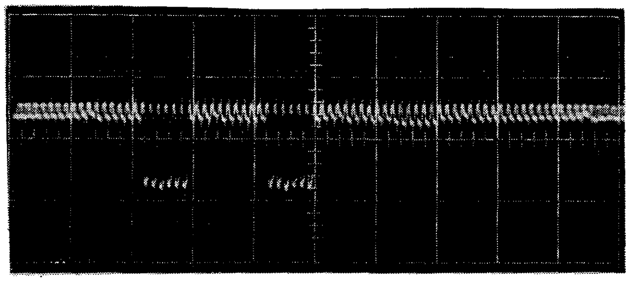


Figure 13—Detected waveforms for two 1024 bit memory cell connected in series by means of an on-the-chip signal regeneration stage

The packing density of CCD memories can be high because only a few MOS capacitors are required to store one bit. For example, using two-phase polysilicon-aluminum gate structures with "standard" layout rules (.2 mil gap, .1 mil alignment), and 1.0 mil wide channels, areas per bit of 1.2 mil^2 are possible. This offers an advantage of 2 to 4 over present dynamic MOS memories.[15] However, smaller areas per bit are likely. Wegener has discussed probable areas per bit for various CCD technologies assuming .5 mil wide channels and line widths of .2 mil, gaps of .2 mil and alignment of .05 mil.[23] With these assumptions, areas per bit range between .3 and .8 mil^2, depending upon the technology used, with .5 mil^2/bit achievable with four different CCD technologies.

Augusta and Harroun point out that for a four-phase serpentine structure the area per bit can be expressed as 15.4W^2 where W is the minimal line width.[24] They also predict that line widths of .15 mil will be possible on a production basis in the late seventies resulting in a CCD area per bit of .35 mil^2.

These predictions all assume essentially conventional photolithographic processing, which is the factor which limits cell size. However, CCD's appear capable of going to lower areas before signal-to-noise ratios become too small for practical operation and therefore the exploitation of high resolution electron beam definition in CCD fabrication may result in even lower areas per bit.

The practicality of going to larger chip sizes because of simpler CCD processing is difficult to assess. The full potential of charge-coupled imagers will only be achieved with chip sizes approaching 500 mils on a side. A large industry-wide effort is being directed—and rather successfully—to this end. It is therefore reasonable to expect that CCD memories can also achieve larger chip sizes than 150 mil. Assuming a 250 mil chip and .5 mil^2 area per bit, 125,000 bits per chip should eventually be possible.

This larger number of bits per chip will result in lower cost per bit. At present CCD memories can probably be manufactured for 0.1¢/bit. To achieve a more attractive cost of 0.02¢/bit or less, the full potential of CCD memories for high packing density and large area chips must be demonstrated with high yield.

Data rate

The analysis of the transfer of the free charge indicates that the maximum bit rate for surface channel CCD serial memories will be approximately 10^7 bits/s. The chip selection and other signal switching may, however, impose a practical upper limit on the bit rate in the range between 10^6 to 10^7 bits/s. The achievement of a high overall bit rate may be achieved, of course, by a suitable multiplexing arrangement.

Power requirements

The power consumed in a CCD memory (aside from regeneration and I/O stages) is essentially that required to charge the storage capacitances. This power will be dissipated mostly in the clock drivers—not on the CCD chip itself. Each bit requires CV^2f power. Thus the total power necessary for a serpentine system will be $CV^2f_cN_b$ where N_b is the total number of bits. The data rate per loop would be f_c bits/sec. Clearly, the power consumption could be reduced by reducing f_c. All unselected loops could be operated at a lower idle frequency with a large reduction in power. Consider, for example, the case of a million bit memory with capacitance per bit of .05 pF (.5 mil^2 at .1 pF per mil^2) and clock voltages of 5 V. The total power required at $f_c = 10$ MHz would be 12.5 watts. If the memory were idled at 10 KHz, the CV^2f power required would be 12.5 milliwatts. The lowest practical idle frequency is limited by thermal generation during the time a particular bit spends between regeneration stages. The time between regeneration stages should be less than .1 to 1 second at room temperature. Thus the lowest idle frequency is related to the number of stages between regeneration (N_R) according to:

$$f_c, \text{idle} \geq .1\ N_R \tag{2}$$

This is appropriate at room temperature. A factor of two increase is required for every 10°C above room temperature. Since frequent regeneration implies low idle frequency and low standby power but decreases the number of storage sites per chip, a basic design trade-off involving power and cost is involved in the selection of the number of stages between regeneration.

The power required for the SPS memory loop is lower than the serpentine system because most of the gates in an SPS system operate at a reduced frequency. For an SPS system with N_x bits across and N_y bits down, the power required is:

$$P_{SPS} = CV^2f_cN_b\left(\frac{2}{N_y} + \frac{1}{N_x}\right) \tag{3}$$

Comparing this with the serpentine loop, assuming a million bit memory made up of SPS loops with $N_x = N_y = 50$, the total power required (at a read-out rate of 10 MHz) would be .75 watts (vs. 12.5 watts for the serpentine equivalent). Only one regeneration stage would be required for every 2500 bits, but each bit would only be transferred 100 times between regenerations. While the CCD memory is a dynamic device and is therefore volatile, the low power required per bit, especially in a standby mode, indicates that rather large memories can be supported on battery power for reasonably long periods of time in the event of primary power failure.

Access time

One drawback of a CCD memory is its serial nature with the associated long access time. The average access time to a random bit is determined by the number of bits between I/O stages and the clock frequency.

$$\tau_{\text{access}} = \frac{N_{I/O}}{2f_c} \tag{4}$$

This trade-off between access time and number of bits between I/O stages is the most significant decision involved in designing a CCD memory. Short access times ($\sim$10 μsec) can be obtained only at the expense of many I/O stages, significant on-chip address decoding and high power consumption because of high clock frequency. The serpentine structure would be required to achieve these low access times. The packing density advantage of CCD memory would be compromised.

At the other extreme, one could utilize only one I/O to a chip minimizing overhead circuitry. Assuming 50,000 bits on a chip and a clock frequency of 10 MHz, the average access time would be 2.5 millisec. A rather wide latitude in access time is possible and no particular structure is optimum for all applications.

Error rate

Since the CCD memory operates by continually transferring charge around a loop, there is some concern about the error rate of such a device. However, analytical[25–27] and experimental[28–29] studies show that the CCD is inherently a low-noise device. When charge transfer is very efficient, little uncertainty can be introduced as to the amount of charge transferred. For long surface channel devices, interface state trapping will limit signal-to-noise ratio, but in memories with a limited number of transfers between refresh, the most important noise source is likely to be associated with the introduction of charge into the first potential well and the sensing and refreshing of the signal. The signal may have shot noise unless special precautions are taken to introduce a low noise signal[28] with only kTC noise. Even if the signal has shot noise, a 0.3 mil^2 gate can store about 10^6 carriers per well which results in a signal power to noise power ratio of 60 db. The error rate introduced with this signal-to-noise ratio would be infinitesimal.

Thus the CCD in itself introduces no special error rate problems—at least not for the size device presently contemplated. Eventually, of course, temporal noise sources may limit the maximum possible packing density.

The error rate of CCD memories will most likely be determined by the characteristics of the refresh stage. Special dynamic detection schemes have been proposed[30] to minimize the error rates at refresh stages due to transfer inefficiency. Other problems which may lead to errors are dark current non-uniformities and threshold variations. The resulting error rates are not expected to be any more serious than those encountered in conventional MOS memory. But because of the serial nature of the CCD memory, such memory systems should be designed with some provisions for error correction.[24]

Typical designs

Agusta and Harroun have presented the conceptual design of an eight megabyte high performance CCD storage device intended for head per track disk replacement.[24] Average access times of 0.5 msec and data transfer rates of 3×10^6 bytes/sec were desired. A serpentine structure with 32,769 bits per chip, with 64 bits between regeneration, 256 bits between I/O (128 on a chip) and a clock frequency of 500 KHz was chosen. The total operating power was expected to be 237 watts with about 43 watts required during standby. An overall volume including control and logic circuitry but not power supplies, of 2.9 cubic feet is anticipated.

Other designs involving SPS loops rather than serpentine loops and fewer I/O stages per chip will result in the lower power requirements and longer access times. For example a chip with sixteen 32×64 SPS loops would also result in 32,768 bits per chip. If the read-out frequency were 2 MHz, the average access time would be .5 msec. On-chip access decoding would be simpler (16 I/O stages per chip vs. 128). Also, the power requirements would be lower, especially if an idle mode were also incorporated within the SPS memory.

DISCUSSION AND CONCLUSIONS

CCD's are at the stage of development where a memory system can be designed—all of the necessary design information is available. However, the question as to whether such CCD memories, if built, would find wide-spread use in general computing machinery depends upon whether they will be cheap enough to compete with existing alternatives—drums or discs and other semiconductor memories. For some applications where the longer access time is of no consequence and certain special features, such as low power dissipation, are very important, CCD memories will very likely be used.

Certainly, CCD memories must show a clear-cut cost advantage over more conventional high packing density semiconductor memories, such as the one-transistor-per-bit store, to be seriously considered for widespread memory system use.

However, the optimism behind the development of CCD memories is based upon their inherent capability for achieving high packing density and good yield on large areas of silicon which should lead to a much lower cost per bit than other semiconductor memories.

REFERENCES

1. Boyle, W. S. and G. E. Smith, "Charge-Coupled Semiconductor Devices," *Bell System Tech. J.*, 587, 1970.
2. Kosonocky, W. F. and J. E. Carnes, "Charge-Coupled Digital Circuits," *IEEE J. Solid-State Circuits*, SC-6, 314, 1971.
3. Engeler, W. E., J. J. Tiemann, and R. D. Baertsch, "Surface Charge Transport in Silicon," *Appl. Phys. Letters* 17, 469, 1970.
4. Kim, C. K. and M. Lenzlinger, "Charge Transfer in Charge-Coupled Devices," *J. Appl. Phys.*, 42, 3586, 1971.
5. Carnes, J. E., W. F. Kosonocky and E. G. Ramberg, "Drift-Aiding Fringing Fields in Charge-Coupled Devices," *IEEE J. Solid State Circuits* SC-6, 322, 1971.
6. Carnes, J. E., W. F. Kosonocky and E. G. Ramberg, "Free Charge Transfer in Charge-Coupled Devices," *IEEE Trans. on Electron Devices*, ED-19, No. 6, June 1972, p. 798.
7. Carnes, J. E. and W. F. Kosonocky, "Fast Interface States Losses in Charge-Coupled Devices," *Appl. Phys. Ltrs.* 20, 7, April 1, 1972, p. 261.
8. Walden, R. H., R. H. Krambeck, R. J. Strain, J. McKenna, N. L. Schryer, and G. E. Smith, "The Buried Channel Charge-Coupled Device," *Bell Syst. Tech. Journal*, 51, 1635, 1972.
9. Esser, L. J. M., "The Peristaltic Charge-Coupled Device," paper presented at the *Charge-Coupled Device Applications Conference*, Sept. 1973, San Diego, California.
10. Amelio, G. F., M. F. Tompsett and G. E. Smith, "Experimental Verification of the Charge Coupled Device Concept," *Bell System Tech. Jour.*, Briefs, p. 593, April 1970.
11. Kosonocky, W. F., and J. E. Carnes, "Charge-Coupled Digital Circuits," *Digest of Technical Papers IEEE Solid State Circuit Conference*, p. 162, Feb. 19, 1971.
12. Kosonocky, W. F. and J. E. Carnes, "Two-Phase Charge-Coupled Devices with Overlapping Polysilicon and Aluminum Gates," *RCA Review*, 34, No. 1, p. 164, March 1973.
13. Kim, C-K. and E. H. Snow, "P-channel Charge-Coupled Device with Resistive Gate Structure," *Appl. Phys. Ltrs.*, 20, 514, 1972.
14. Collins, D. R., S. R. Shortes, W. R. McMahon, T. C. Penn, and R. C. Bracken, "Double Level Anodized Aluminum CCD," Paper presented at *IEEE Int'l Electron Devices Meeting*, Washington, D.C., Dec. 1972.
15. Krambeck, R. H., R. J. Strain, G. E. Smith, and K. A. Pickar, "A Conductively Connected Charge-Coupled Device," Paper presented at *IEEE Int'l. Electron Devices Meeting*, Washington, D.C., December 1972.
16. Kosonocky, W. F. and J. E. Carnes, "Design and Performance of Two-Phase Charge-Coupled Devices with Overlapping Polysilicon and Aluminum Gates," *1973 International Electron Devices Meeting Technical Digest*, p. 123, Washington, D.C.
17. Krambeck, R. H., R. H. Walden and K. A. Pickar, "Implanted Barrier Two-Phase CCD," *International Electron Devices Meeting*, Washington, D.C., October 11-13, 1971.
18. Gunsagar, K. C., C. K. Kim, and J. D. Phillips, "Performance and

Operation of Buried Channel Charge-Coupled Devices," *1973 IEEE Int'l. Electron Devices Meeting Technical Digest*, p. 21, Washington, D.C.

19. Erb, D. M., W. Kotyczka, S. C. Su, C. Wang, and G. Clough, "An Overlapping Electrode Buried Channel CCD," *1973 IEEE Int'l. Electron Devices Meeting Technical Digest*, p. 24, Washington, D. C.
20. Tompsett, M. F., "The Quantitative Effects of Interface States on the Performance of CCD's," *IEEE Trans. on Electron Devices*, ED-20, No. 1, 45, 1973.
21. Collins, D. R., J. B. Barton, D. D. Buss, A. R. Kmetz and J. E. Schroeder, "CCD Memory Options," *1973 IEEE Int'l. Solid-State Circuits Conference*, Digest of Technical Papers, p. 136, Philadelphia, Pa.
22. Chan, Y. T., B. T. French and R. A. Dugmundsen, "Charge-coupled-memory device," *Appl. Phys. Lett.*, 22, 1973.
23. Wegener, H. A. R., "Appraisal of Charge Transfer Technologies for Peripheral Memory Applications," *CCD Applications Conference Proceedings*, p. 43, San Diego, California, September 1973.
24. Agusta, B. and T. V. Harroun, "Conceptual Design of an Eight-Megabyte High Performance Charge-Coupled Storage Device," *CCD Applications Conference Proceedings*, p. 55, San Diego, California, Sept. 1973.
25. Barbe, D. F., "Noise and Distortion Consideration in CCD's," *Electronic Letters*, 8, 207, 1972.
26. Carnes, J. E., and W. F. Kosonocky, "Noise Sources in Charge-Coupled Devices," *RCA Review*, 2, 327, 1972.
27. Thornber, K. K., "Noise Suppression in Charge-Transfer Devices," *Proc. IEEE* 60, 1113, 1972.
28. Carnes, J. E., W. F. Kosonocky, and P. A. Levine, "Measurements of Noise in Charge-Coupled Devices," *RCA Review*, 34, 4, December 1973.
29. Emmons, S. P. and D. D. Buss, "The Performance of CCD's in Signal Processing at Low Signal Levels," *CCD Applications Conference Proceedings*, p. 189, San Diego, California, September 1973.
30. Thornber, K. K., "Operational Limitations of Charge Transfer Devices," *Bell System Tech. J.*, 52, 9, p. 1453, November 1973.

Block-oriented random access MNOS memory

by J. E. BREWER

Westinghouse Electric Corporation
Baltimore, Maryland

and

D. R. HADDEN, JR.

US Army ECOM

INTRODUCTION

MNOS storage elements have been used to realize a block-oriented all electronic secondary memory module. This nonvolatile storage unit offers 10-microsecond data access and reliable error free operation. BORAM modules provide an immediately available cost effective alternative to electromechanical storage in severe environment applications. The text below describes an 18-million-bit advanced development model of a BORAM module and briefly outlines the growth potential of MNOS BORAM systems.

BORAM CONCEPT

The acronym BORAM and the concept of a block-oriented memory were originated at ECOM in June of 1963. Motivated by frustration with the characteristics of available secondary storage, ECOM personnel drew up a set of desired performance goals. It was reasoned that if data were manipulated in blocks, significant simplification in addressing and control circuitry could be achieved. BORAM was constrained to an all electronic implementation, but otherwise no particular storage technology was specified.

During the next ten years ECOM diligently sought a practical realization of the BORAM. The Electronics Command worked with several contractors to explore the feasibility of magneto-sonics, ferro-electrics, traveling domain walls, magnetic wires, bubbles, electron beams, and MNOS. Of these options only the MNOS approach was confirmed to be currently ready for production.

A BORAM module can be configured for many different data structures. In the advanced development model information to be stored is processed in bytes consisting of 8 data bits and 1 parity bit. A block is defined to be a sequentially ordered set of 2048 bytes. The BORAM module can store 1024 blocks. Any block can be addressed and written or read. An operation (read or write) will always begin with the first byte of the addressed block, and then will proceed to the higher order bytes in sequence.

MODULE FUNCTIONAL STRUCTURE

Figure 1 identifies the six functional parts of the BORAM module. The storage section provides nonvolatile read/write storage for 18,874,368 bits. It contains MNOS memory chips, associated drivers and buffers, and power switching circuitry.

The I/O section performs all communication with the external controller. It accepts control signals and data, and outputs status signals and data. Internal to the module it issues commands to the data buffer and control section, and transmits the block address to the block selection section.

An important aspect of the module design is that it can be used with different computer systems simply by insertion of a different I/O card. At the heart of the I/O section is a small microprogrammed controller. Changes in microprogram customize the I/O logic for a particular installation. Other custom elements such as the line drivers and receivers also appear on this card.

A small data buffer and the module control circuitry are located on one PC card. The control circuitry responds to requests from the I/O section and provides signals to operate the storage section and the data buffer. This circuitry also generates all timing signals and data bits for error detection purposes.

Dynamic shift registers are employed within the BORAM storage chips for data input and output. Synchronous transfer of data between the buffer and the storage section avoids any possible loss of data. Transfers between the buffer and I/O are asynchronous and long delays may be tolerated. The buffer also performs a data format conversion

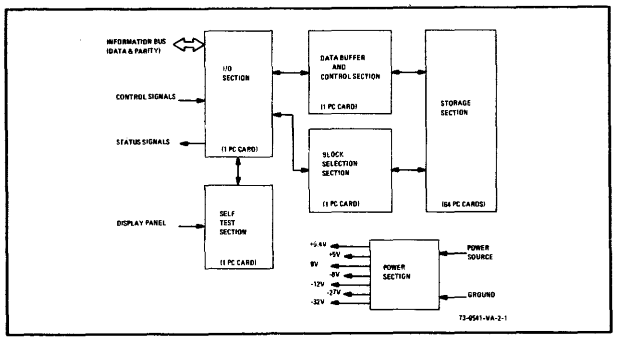


Figure 1—BORAM module functional block diagram

function. Data enters the BORAM module in 9-bit bytes and is converted to an 18-bit format as it enters the storage section.

The block selection section accepts a block address from the I/O section and generates the signals which enable the appropriate drivers and memory chips in the storage section.

Self-test electronics are powered down during normal module operation. A front panel switch and readout allow manual confirmation of the module performance.

PHYSICAL STRUCTURE

The BORAM module is intended to be a general purpose storage unit for military applications. Ground fixed, airborne inhabited, and ground mobile operating, environments are specified. The operating ambient temperature range is −46°C to +52°C. Storage temperatures can vary from −57°C to +71°C.

As shown in Figure 2 the module is housed in a rugged case. It can be mounted and used in the case, or the rear portion of the case can be removed and the module can be mounted on chassis slides.

The front panel fin structure serves to transfer heat to the external environment and also provides a mounting surface for the module controls and indicators. As indicated in Table I, the unit weighs 120 pounds and occupies 4 cubic feet.

STORAGE ELEMENT

The BORAM storage element is a drain-source protected MNOS memory transistor (see Reference 1). This Westinghouse proprietary transistor structure avoids the read window closure degradation that is characteristic of nonprotected MNOS transistor designs.

Nonprotected memory transistors generally become unusable after about 10^6 clear/write cycles. Tests on drain-source protected transistors are still in progress and have exceeded 10^{12} clear/write cycles. No significant read window closure has been observed.

Elimination of degradation is an essential prerequisite for application of MNOS in BORAM systems. Figure 3

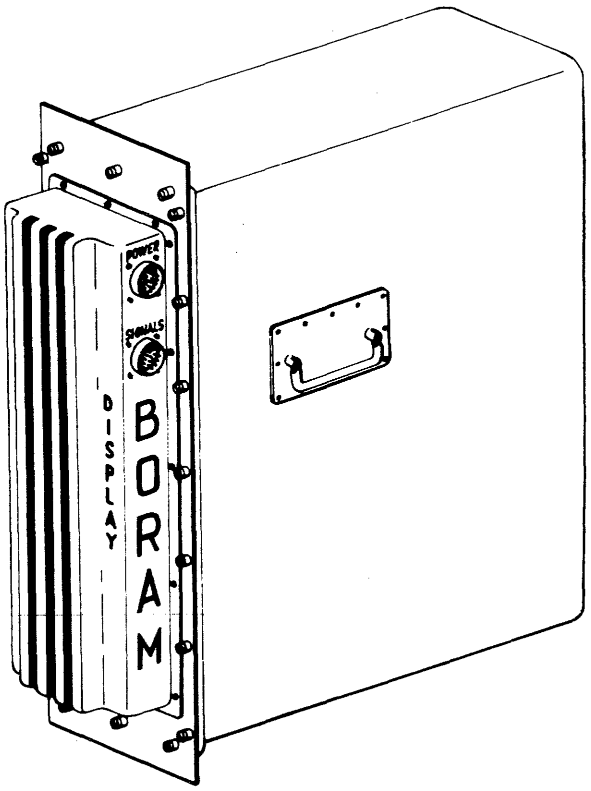


Figure 2—BORAM module housed in transit case

shows the wide range of cycling which can be experienced by a memory transistor within a BORAM. The plot assumes the operating parameters specified for the advanced development model, and operating time is defined as the actual time the module is engaged in a read or write operation. Obviously a 10^6 cycle limitation could not provide a satisfactory system operating life.

STORAGE SECTION

Data is stored in memory chips which have been designated as "BORAM 6000" integrated circuits (see Reference

TABLE I—BORAM Module Characteristics

CHARACTERISTIC	MAGNITUDE	UNITS
STORAGE CAPACITY	18,874,368	BITS
READ ACCESS TIME	10	MICROSECONDS
DATA TRANSFER RATE	2X10^6	BYTES/SEC
MODULE POWER	100	WATTS
VOLUME	4	CUBIC FEET
WEIGHT	120	POUNDS

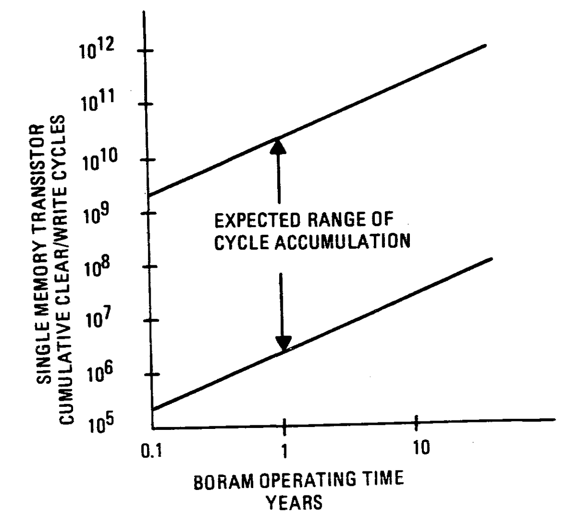


Figure 3—Clear/write cycle accumulation for a memory transistor in a BORAM 6000 chip under normal BORAM module operating conditions

2). Each chip provides 2048 bits of nonvolatile, nondestructive readout storage. The chip contains a fully decoded RAM organized as 64 words by 32 bits, and two 16-bit two-phase dynamic shift registers. All data I/O takes place serially via the shift registers. This scheme considerably reduces the operating speed requirements imposed on the RAM.

The memory chip features a read access time of about 5 microseconds and a register shift rate of 3.3 megahertz over the operating temperature range. For the BORAM module use conditions retention time exceeds 4000 hours.

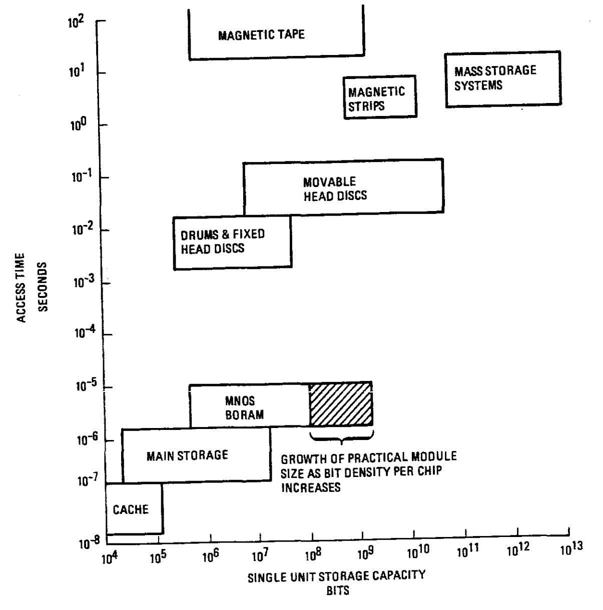


Figure 4—Approximate access time and capacity for various classifications of storage

TABLE II—Typical 1974 Military System Secondary Storage Technology Options

TECHNOLOGY	ACCESS TIME Microseconds	DATA RATE Megabytes/Sec	VOLUME Bits/Cubic Inch	WEIGHT Kilobits/Pound	POWER Microwatts/Bit	SYSTEM PRICE Cents/Bit
PLATED WIRE	0.4	2.6	645	24	20	8
MAGNETIC CORE	1.2	1.4	503	25	55	3
BORAM (MNOS)	10	2.0	2730	157	5	2
FIXED HEAD DRUM OR DISC	$9X10^3$	0.2	671	42	18	0.5
MOVABLE HEAD DISK	$2X10^5$	0.08	1956	153	7	0.1

The primary building block of the storage section is a hybrid microcircuit which contains 16 of the BORAM 6000 chips. This package will store 32,768 bits. Nine microcircuits are placed on a printed circuit card with associated driver, buffer, and power switching circuitry.

Power switching is particularly effective in the storage section. When the section is engaged in a read or write, approximately 10 watts are dissipated. When the module is active, but the storage section is not engaged in a read or write, about 1.7 watts (<0.1 microwatts/bit) are dissipated. The bulk of module power is required for functions outside the storage section.

NONVOLATILE TECHNOLOGY COMPARISON

Because MNOS BORAM is a new technology, it seems worthwhile to try to position it approximately within the context of other technology options. Figure 4 shows that BORAM offers about three orders of magnitude improvement in access time over fixed head drums and discs. At present it appears practical to build BORAM modules as large as 10^8 bits. This exceeds the capacity of fixed head systems and encroaches on many moving head designs.

Table II contrasts the characteristics of some specific existing military storage systems against the BORAM advanced development models. Obviously these systems constitute only one sample of what a technology can achieve, and this data should be treated accordingly. Parameter computations encompassed total operating configuration including such items as drives, controllers and power supplies.

Slow versions of plated wire and core can be used to configure secondary storage with faster access time than the BORAM, but serious penalties would be incurred in cost, volume, weight, and power. In every category except cost the BORAM offers advantages over fixed head drum and disc configurations. The moving head system enjoys more than an order of magnitude cost advantage, but the BORAM has equivalent or better characteristics in the other categories.

Comparison of the reliability of these alternatives is a more complex task. Reliability is a function of the use environment. For rugged environments like the ground mobile application electromechanical systems should be avoided.

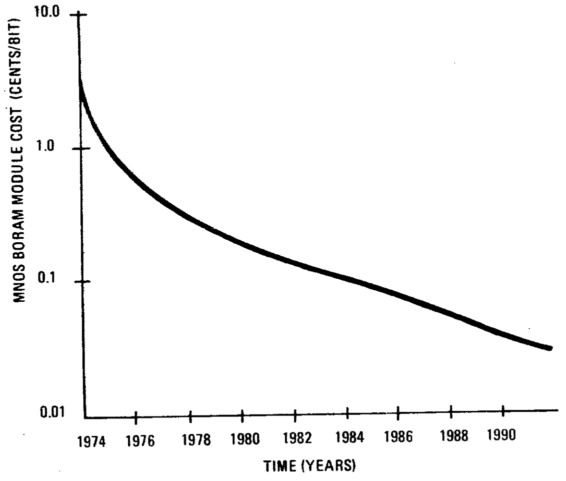


Figure 5—Expected cost trend for BORAM modules

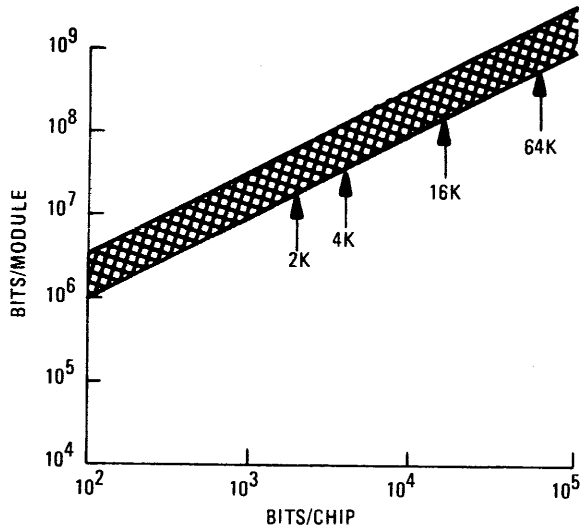


Figure 6—Practical BORAM module capacity without error correction and/or redundancy as a function of chip bit density

Here BORAM is the most cost effective all electronic implementation option.

In mild environments the BORAM is expected to have about twice the MTBF of a high reliability fixed head system of the same capacity. Because of the newness of the MNOS BORAM this expectation has not yet been verified by actual test.

MNOS BORAM GROWTH POTENTIAL

As the MNOS technology matures significant reductions will occur in cost/bit, and the storage capacity of a single BORAM module will increase. These changes will be brought about by the usual "learning curve" improvements associated with integrated circuits and by the introduction of new memory chips with higher bit density.

Figure 5 presents a conservative projection of the trend for module selling prices. By 1978 MNOS BORAM modules will have a cost advantage over fixed head disc/drum systems for military applications. Eventually the MNOS BORAM will become cost competitive with militarized moving head disc systems.

From reliability considerations it can be shown that a practical limit to BORAM module size is 10,000 to 30,000 memory chips. If error correction and/or redundancy is incorporated in the module, even larger configurations become practical. The nonvolatility of the memory chip allows the bulk of the system to remain powered down, and thus the chip failure rate is reduced to that of a nonoperating chip.

Figure 6 shows how the expected increases in bit density per chip will affect module storage capacity. In 1974 a 4K chip will be available, and 100-megabit modules will be feasible. Eventual growth to 1000-megabit modules is within the capability of the technology.

SUMMARY

Block-oriented random access memory has evolved from its initial conception at ECOM as a set of idealized performance requirements to a practical working model. This 18-megabit module was implemented using MNOS memory chips designed especially for the BORAM application. Module organization and circuit design has achieved a flexible high utility configuration suitable for use in rugged environments.

MNOS BORAM provides advantages in access time, size, weight, power and reliability when compared against fixed head electromechanical systems. At present MNOS BORAM is a cost effective alternative to electromechanical memories in severe environment applications. In the near future BORAM will compete with a large class of electromechanical memories on a direct cost basis.

REFERENCES

1. Cricchi, J. R., F. C. Blaha and M. D. Fitzpatrick, "The Drain Source Protected MNOS Memory Device and Memory Endurance," *IEEE International Electron Devices Meeting*, December 1973.
2. Cricchi, J. R., J. E. Brewer, D. W. Williams, F. Blaha and M. D. Fitzpatrick, "Nonvolatile Block Oriented Random Access Memory," *IEEE 1974—Solid State Circuits Conference.*

DOT memory systems

by R. J. SPAIN, H. I. JAUVTIS and F. T. DUBEN

Cambridge Memories, Inc.
Newton, Massachusetts

INTRODUCTION

New developments in memory technology are attempting to fill the void between main and secondary storage by providing alternatives to rotating storage devices with considerable performance advantages over the latter and at far less cost than main memory. The greater emphasis being placed on the use of virtual storage techniques in data processing systems has reaffirmed the need for fast, reliable, low cost memory devices with much reduced page swapping times. A principal developmental effort in this area is the development of moving magnetic domain techniques. The DOT, or Domain Tip, storage technology has an intrinsic capability for providing non-volatile, fast access, block or page-oriented memory systems. The relatively few steps required to manufacture DOT devices and the simple planar packages used offer the potential of low manufacturing cost. The memory medium, insensitive to temperature and to shock and vibration, is electronically passive with no connections being required between the storage medium and the drive or sense electronics. These factors suggest the capability of the DOT of providing highly reliable solid state memory.

In principle, the DOT stores and retrieves information in a manner similar to that of rotating magnetic storage media, that is, by storing information in the form of a moving pattern of magnetic domains. However, by not having the mechanical constraints of a disk or drum, significant advantages are possible. For one, the data tracks are easily segmented into minor storage loops so as to reduce the transport time for retrieving or storing data. By being able to organize the memory such that pages of data are electronically accessed, extremely short latency times can be provided. This same block organization, in addition to the ability of the DOT to provide extremely high signal to noise ratio, assures that the associated system electronics is minimal in amount and in cost.

This paper will review the DOT technology and describe the devices and memory substrate organizations suitable for two specific mass memory systems. The first is a 2.5 million bit prototype memory system having a data rate of 240 kilowords per second. The second is a larger capacity paging store having a total capacity of 24 million bits and a data rate of 1 megabyte per second. The smaller memory has been built and lifetested. The second, larger one is presently under construction. The characteristics of these two systems are described and the tests results obtained to date are presented.

DOT TECHNOLOGY

The DOT makes use of a magnetic storage medium within which highly mobile, information-bearing magnetic domains can be electronically shifted along a prescribed set of batch-fabricatable channels or tracks. Information is entered into a DOT memory substrate, shifted along the channels and recirculated or read out by a combination of magnetic fields produced by an associated conductor pattern and drive currents.

The DOT propagation channels are produced by constructing a memory substrate such that at all points external to these channels a particularly high threshold field for domain wall motion exists. A significantly lower threshold field is retained only within the narrow propagation channels. In this manner, application of the magnetic drive fields which operate the DOT device can cause domain wall motion only within the channels; no switching of the magnetization occurring elsewhere. The narrowness of the channels and the intrinsic magnetic anisotropy of the storage material make it possible to move domain walls within the channels at fields which are far less than that required to spontaneously create new domains. As a result, only those domains expressly introduced into the channel pattern for the purpose of storing or processing digital information will be retained or propagated in the course of operating the device.

In order to achieve the channel and magnetic film properties described above, several fabrication methods have been developed which make use of an under or over-layer to the magnetic storage film for selectively producing the desired variations in domain wall coercive force in a manner compatible with conventional photolithographic techniques. A preferred technique uses an underlayer of aluminum in the form of a thin, superficially rough film. This film is photoetched so as to leave aluminum only at points external to what are to become the propagation channels once the memory substrate is completed. At present bit densities, a 0.0015″ wide channel is defined at this point. Completion of the memory substrate, following this photoetching step, requires depositing the magnetic storage layer as a uniform, continuous film across the entire memory plane surface.

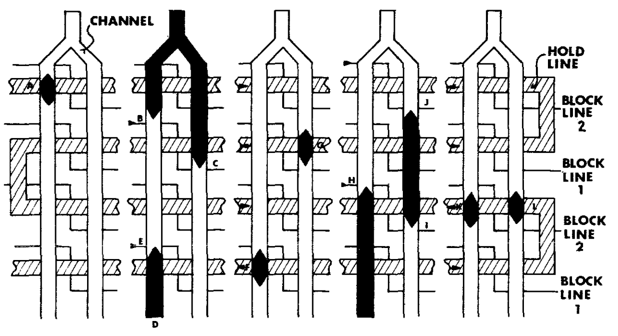


Figure 1—Domain pattern during shift cycle in blocking type shift register

The channel structure outlined above requires very few steps in its manufacture, making it possible to achieve high device yield and very low cost. The aluminum and magnetic layers are deposited by vacuum deposition, the aluminum film to a thickness of approximately 300Å and the magnetic film to a thickness of roughly 1000Å. The latter is a NiFeCo layer, the Co content of which allows a large magnetic anisotropy and consequentially a high domain nucleation threshold to be obtained. The NiFe ratio is chosen such as to assure stress insensitivity. This condition keeps the domain wall coercive force within the channel relatively low and eliminates any stress-induced changes in the device from occurring either in the course of device assembly or during operation under conditions of shock and vibration.

DOT SHIFT REGISTER DEVICES

Information, in the form of a sequence of magnetic domains, can be propagated through the DOT channel paths in several different ways. One technique uses a zigzag channel pattern and the dependence of the angle of tip propagation on the angle of the applied field to cause the unidirectional motion of the stored domain pattern. A second technique uses a DOT magnetic diode element to accomplish the same result. A third and preferred technique and the one which has been used in all recent DOT memory systems is the Blocking Type shift register.

The basic device consists of a DOT memory substrate positioned on a control conductor pattern and contained within a flat, wrap-around solenoidal drive coil. Information, in the form of a sequence of domains of reversed magnetization, is entered into the channel structure of the DOT memory substrate by means of an input conductor which is used to create new domains at an input location. These domains are shifted through the channels by a sequence of uniform drive and local control fields and readout at the device output by a sense conductor into which a signal is induced by the passage of the output domain. The motion of domains through the pattern of channels resembles the movement of a group of inchworms in that the domains are made to periodically expand and contract, expansion being greatest at the leading tip of the domain and contraction being greatest at the trailing tip. Expansion and contraction are produced by the propagation and erase fields generated by the wrap-around drive coil.

The blocking type shift register utilizes a simple pattern of straight channels within which domain expansion is controlled by the magnetic fields produced by a set of conductors called block lines. The channel merely functions as the information transport medium while the direction of domain propagation is determined by the spatially varying resultant of the uniform propagate and local blocking fields. Hence, the operating margins of the shift register are relatively independent of variations in channel geometry and film properties, making high fabrication yields possible. Substrate yields of thirty percent are typical and yields of over ninety percent have been obtained.

The operation of the blocking type shift register is illustrated in Figure 1 which depicts the domain pattern during the shifting sequence. The hold conductor, whose intersections with the channels define the local storage locations, runs perpendicular to the channels and alternates its direction across the pattern. The two block lines, block 1 and block 2, are contained on a second conductor layer and carry current in the same direction at all points where they intersect the channels. These lines are configured in a manner which makes it possible to alternate the direction of domain propagation from channel to channel. This allows the propagation paths to be folded back and forth and configured in whatever manner is desired.

The shift sequence takes place as follows: The initial state of the magnetization is shown in Figure 1a with a domain stored at location A. The cycle starts with propagate and block 1 pulses occurring simulataneously. The propagate field is applied uniformly over the entire memory substrate in the direction shown while the field from block line 1 opposes propagation field at points B, C and E. Since the block field is adjusted to cancel the propagate field, domain expansion is inhibited at these points. New information is seen entering the left channel at D. An erase and hold operation occurs next and domains of reversed magnetization are stored at locations F and G while all other channels are erased as illustrated in part C of the figure. The third step in the sequence is another propagate pulse during which the block line 2 is energized. As a result, the domain tip at F is blocked at H and the one at G comes to rest at I and at J as shown. The shifting sequence is completed by a second erase and hold operation with the polarity of the hold current reversed. Domain contraction now occurs about points K and L which become the storage locations at the end of the shift cycle.

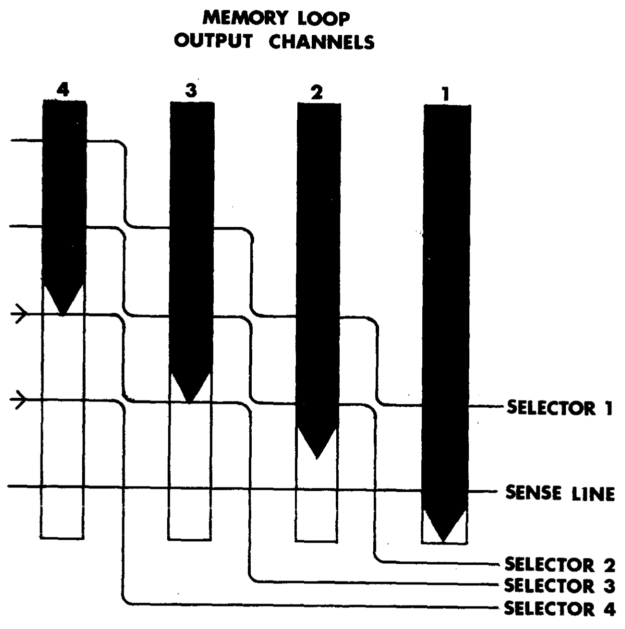
Either interchanging the block lines or reversing the polarity of the hold pulses will reverse the direction of information propagation. This feature can be used to reduce the access time to a particular location in the case of a closed loop, magnetically recirculating channel path.

Typical operating margins are relatively large. In general, variations in all drive currents of plus and minus twenty percent are allowable, providing sufficient operating margin for ease of system design and construction.

Current bit densities for use in DOT systems have not yet exceeded 6000 bits per square inch in order to assure high device yield and to minimize mechanical problems in device test and plane assembly. Shift registers of this type, at densities of 32000 bits per square inch have been operated as well but the tighter mechanical tolerances involved and the reduced substrate yield have, to date, discouraged their use to any large extent in actual operating systems.

The velocity of domain tip propagation is a function of the applied magnetic field strength, increasing as the second or third power of the field. In the range from 5.5 to 8.5 oe., the normal allowable propagation field variation, the velocity of propagation lies between 10 and 20 mils per microsecond. At a density of 6000 bits per square inch the total distance a domain must travel is approximately 30 mils. At 15 mils per microsecond a total propagation time of 2 microseconds is necessary. Domain erasure is much more rapid.

While a 2 microsecond cycle time and consequently a 1 megacycle channel data rate is achievable and while, by increasing drive field strength or longitudinal bit density, even shorter cycle times are possible, these frequencies of operation are not conveniently attained at a system level using current packaging techniques and signal level requirements. Firstly, domain expansion for flux amplification at the readout location can take up to 10 microseconds in some of the designs used to date. Whereas extremely small and higher speed magneto-resistance DOT readout elements have already been developed, the simplicity of assembly and the greater potential reliability of an inductive sensing approach have discouraged the introduction of this sensing technique. Secondly, the relatively large inductance (6 microhenries) of each wrap-around drive coil and the requirement for overlap of certain drive currents limits the frequency of device operation for purely electronic reasons. Providing rise and fall times of one microsecond and allowing ample separation between edges of overlapping currents to eliminate precise control over pulse timing result in cycle times of 20 microseconds. Eventually, use of non-inductive sensing techniques and a more advanced package with lower inductance drive lines will permit the intrinsic speed of the magnetic element to be more closely approached. For the moment, high data transfer rates can nevertheless be obtained by the use of simple multiplexing techniques, parallelism or a combination of both.



(a)

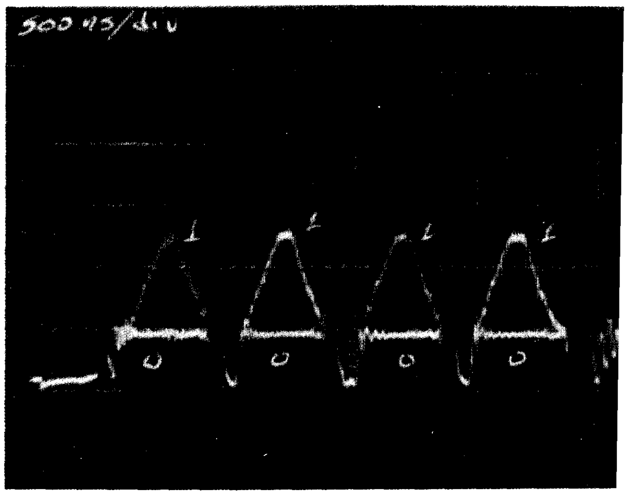


Figure 2—Time multiplexing principle (a) Output domain from memory track 2 is shown approaching sense line following turn off of selector 2 current, track 1 has already been read; outputs from tracks 3 and 4 have moved one position closer to readout (b) Multiplexed output signals (500 ns per div.)

MEMORY SUBSTRATE ORGANIZATION

Simple magnetic selection methods make it possible to straightforwardly multiplex a multiplicity of relatively short DOT memory loops onto a common input or output data line. An important part of memory system design is to be able to reduce the length of the longest data path of a propagated domain through the memory while simultaneously retaining a large number of data bits per sense and digit line. Designing the memory substrate such that several memory loops can be controllably coupled and decoupled to a single data line can be done in a variety of ways.

One method which has been implemented in a 2.5 million bit prototype system and which appears particularly suited for a memory system in this capacity range is to time multiplex in rapid sequential fashion the input and output signals of a number of magnetic memory loops. All memory loops under the general drive coil undergo domain shifting simultaneously. By producing the blocking of domain tip propagation at the output channel of all such memory loops with the aid of a series of blocking or selector lines, all memory loops become uncoupled from the sense line. This is shown in Figure 2a. Turning the current off in a first selector line allows the output domain from one of the memory loops

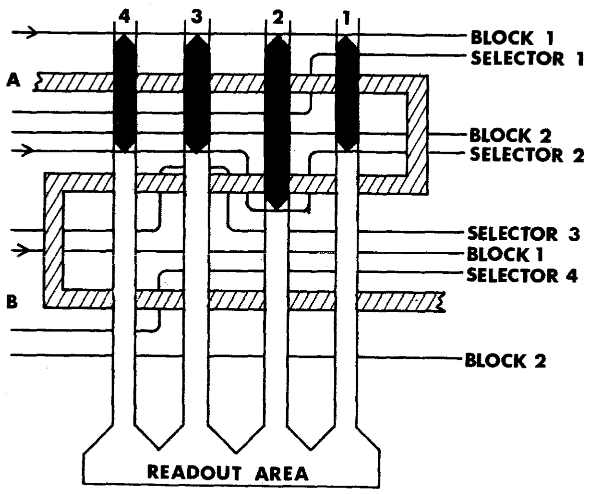


Figure 3—Page multiplexing principle. Output domains from loops 1, 3, 4 are shown inhibited from propagating from A to B by current in selector line 2. Selectors 1 and 2, like block 1, are used during first half of shift cycle; selectors 3 and 4, like block 2, are used during second half of shift cycle

to propagate into the readout area and induce an output signal on the sense line. All output domains from the remaining memory loops move one position ahead. Turning current off in a second selector line allows a second output signal to be read and again advances the remaining output domains. This process continues until signals have been read from all memory loops. This multiplexing process is equally applicable to the inputting of domains. Inputting is done in a corresponding sequential manner by loading a "one" (domain) into the input channel of all memory loops. An identical arrangement of selector lines makes it possible for an inhibit conductor to sequentially inhibit the propagation of the loaded domains past each of the selector lines depending upon the state of the inhibit line, i.e., whether it carries current or not at the moment the appropriate selector line carries no current. The state of the inhibit line at this time is gated by the input data as would be the case in a typical core memory. If magnetically recirculating memory loops are used, the selection and inhibit lines must include the recirculating portion of the memory loops as well.

Time multiplexing in this manner increases the effective data rate by allowing several signals to be read or written in a cycle only slightly longer than the basic shifting cycle. Figure 2b shows output signals being read at a one megacycle rate.

In larger capacity DOT memory systems a different type of multiplexing appears more appropriate. In the organization described above, each memory loop had assigned to it a separate readout area, although all readout areas of the multiplexed loops are in common with the single input output line. If, instead, all multiplexed loops are connected to a common readout area, much larger output signals can be obtained. This, of course, allows a larger number of sense lines to be grouped in series so as to further reduce sense electronics cost and also minimizes the problems of noise cancellation within the stack.

In the previously described substrate memory loop organization, the multiplexed memory loops form part of a single block of data. In the organization now presented each memory loop must correspond to a different data block. That is, as one memory loop is accessed in the process of reading or writing a block of data, data stored in the remaining memory loops must be magnetically recirculated entirely uncoupled from the readout area.

In this case, the added effective throughput that was obtained in the previous multiplex organization no longer exists. However, in the larger capacity system, the preamplifier electronics which forms part of each memory stack can be used to advantage to obtain sufficiently high throughput on the basis of a much wider word. The advantages of obtaining higher output signal become apparent when one considers the 8 millivolt signal currently obtained with substrates organized in this fashion as compared to the 2 millivolt signals obtained in the prototype system of the former design.

Figure 3 depicts the layout of memory loops corresponding to the situation of several magnetic tracks multiplexing onto a common readout area. Again, only the output portion of the selection network is shown, although the same principle can be extended to domain inputting.

Several dummy bits exist between the output leg of the shift register loops and the readout area. It is during these dummy shift cycles that the selection of the appropriate magnetic stack is made. The output selection lines serve the purpose of uncoupling all unaddressed memory loops while allowing the output domains from the selected loop to proceed on into the readout area. At the input of the memory loops, the input selectors serve the functions of allowing all unselected loops to continue magnetically recirculating the data without interference, inhibiting magnetic recirculation in the selected memory loop, and allowing the newly inputted data to enter only the addressed memory loop. Additional dummy bits are used at the end of the data block to ensure completion of this writing function.

SYSTEM ORGANIZATION AND ELECTRONICS

In order to evaluate the feasibility of applying the DOT technology to bulk storage a 2.5 million bit DOT memory system was designed. The system was organized in the form of 64 blocks of data, each block consisting of 2048 words with 18 bits per word. Of the 18 bit wide word, 16 bits are used for data, one for word parity and another for a marker bit which denotes the end of the data block. This organization was realized using stacks of 16 memory planes, the latter each containing 9,216 bits. Sixteen such stacks complete the system.

The system was designed around the time-sequential multiplexing techniques described in the preceeding section. The memory substrates are configured as three sets of four 256 bit recirculating memory loops, the four loops multiplexed onto a common data line by means of four separate selector lines. Three substrates are used per memory plane so as to obtain nine separate data lines. The 18 bit word is the result of driving two memory planes in parallel.

The organization described above permits 2048 word data blocks to be entered into or retrieved from memory at a maximum data rate of 240 kilowords per second, i.e., over 4 megabits per second. Due to the static nature of the DOT, this data rate can of course be reduced by any amount or stopped or started as required. Access to the first word of any block takes only 7 microseconds. At the maximum data rate the average word access time is 4.2 milliseconds. Mechanically, the system is housed in a 10½″ rack-mountable enclosure which contains the 16 stacks and memory drives, timing and sense circuits. System cooling is provided by six high pressure fans located directly under the enclosure.

The electronic drive circuits for all drive currents are identical in design and are built from a matrixed arrangement of sources and sinks utilizing a common current source. Variations from circuit to circuit is minimized since there is only one current source for each drive current. By turning the current source on last and off first and by allowing all selected lines to settle after selection by source and sink, all control of the waveform is established by the current source. All memory drive currents are between 500 and 800 ma, depending upon the function. Total system power dissipation is nearly linear with data rate, reaching 250 watts for the worst case data pattern at maximum data rate.

The sense lines consist of short photoetched conductor loops which are in close proximity to the readout areas of the

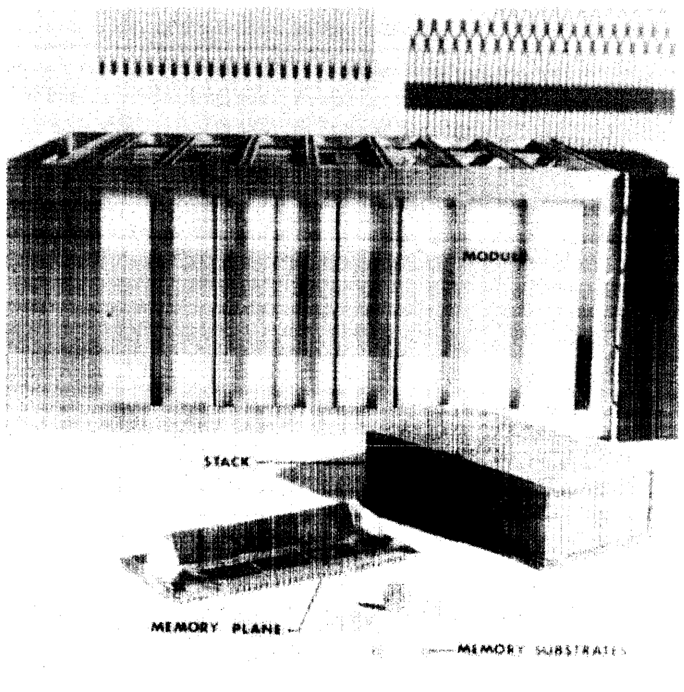


Figure 4—DOT memory components

substrate. Sixteen sense loops in each stack are wired in series for a total of 16K bits per sense line. The output signal from the magnetic film is a bell-shaped 2.5 millivolt signal approximately 0.5 microseconds wide at a half amplitude level. A sense amplifier was designed which is predominantly a dc coupled system so as to minimize the effect of variations in noise recovery resulting from either electrical variations from memory plane to memory plane or due to changes in repetition rate.

This system was constructed and lifetested for well over six months during which time it was possible to verify the fundamental soundness of the magnetic and mechanical design and reliability of the devices. Data, in the form of a worst case repeated 1-0-1-0 pattern, were magnetically recirculated through the memory loops at maximum data rate throughout this period. Temperature of the input air to the system was varied between $-10°$ and $+50°$C. It was confirmed that no degradation in properties of the magnetic elements or package occurred during this lifetesting. The duration of this test allowed error rates of nearly all of the memory loops to be established as less than one error in 10^{12} cycles.

Based upon the results obtained with the prototype DOT memory system described above, a fast access memory for page swapping applications of considerably greater bit capacity was designed. The total system capacity exceeds two megabytes.

The system is designed to make the DOT memory plug compatible with 360/370 processing systems. Organized around a 2K byte page of data, the expected performance is a page access time of 0.2 milliseconds and an effective page transfer time of 2.25 milliseconds. This corresponds to the transfer of 440 kilopages per second.

The storage unit of the system contains 24 memory modules accessed in parallel. Each memory module is complete with drive and sense electronics and four 32K by 10 bit memory stacks. The 20K bits of each memory plane are made using memory substrates configured using the second multiplexing technique described. Eight recirculating magnetic memory loops (or portions of pages) are multiplexed onto a common readout area. This organization results in output signals of 8 mv with nine sense lines per stack and 32K bits per sense line. The first storage modules and interface electronics of this memory system are currently being fabricated and tested.

The memory substrates, planes and stacks of which these systems are constructed, are shown in Figure 4. In the background of the figure can be seen the memory module into which the stacks are inserted as well as an enlarged portion of a portion of a typical DOT channel pattern.

CONCLUSION

The technical feasibility of the DOT storage technology has been demonstrated by means of constructing and lifetesting several prototype memory systems, the largest of which is a 2.5 million bit memory. At present, a prototype system nearly 10 times this capacity is being fabricated. Experience with the smaller system has shown that the DOT can provide highly reliable, fast access storage. The experience obtained on substrates yield and the mechanical packaging techniques developed suggest that this performance can be accompanied by low manufacturing costs. The 24 million bit memory presently under construction will serve to demonstrate what system performance advantages can be achieved using memories of this type. Specifically, in a virtual storage environment the requirement for rapid swapping of pages of data back and forth between secondary and main memory should make the short access time and rapid transport of data in the DOT memory system extremely desirable features.

The first prototype system utilized DOT memory planes having a capacity of 9 kilobits; the system presently under construction utilizes memory planes which have a bit capacity of 20 kilobits. Still higher capacity memory planes are being designed. The simplicity of manufacturing of the storage medium itself and the ease and connection-free assembly of memory substrates to drive conductors promises to keep production costs of these memories low and reliability high. We expect to be able to demonstrate significant improvements in data processing performance using memories of this type. We hope by this to show that non-volatile, electronically addressable page-oriented DOT memory systems are a desirable substitute for rotating storage, particularly in the case of memory hierarchies where the differences between access times of main and secondary memory become intolerably great.

BIBLIOGRAPHY

1. Spain, R. J. and H. I. Jauvtis, "Controlled Domain Tip Propagation," *J. Appl. Phy.*, Vol. 37, No. 7, pp. 2572-2583, June 1966.
2. Spain, R. J., "Domain Tip Propagation Logic, *IEEE Trans. on Magnetics*, Vol. MAG-2, No. 3, pp. 347-351, September, 1966.
3. Jauvtis, H. I. and R. J. Spain, *DTPL All-Magnetic Logic Networks*, presented at the 1968 International Conference on Magnetics, Washington, D.C., April 1968.
4. Spain, R. J. and M. Marino, "Magnetic Film Domain Wall Motion Devices, *IEEE Trans. on Magnetics*, Vol. MAG-6, No. 3, pp. 451-463, September 1970.
5. Jauvtis, H. I., *The DOT Technique of Magnetic Storage and Logic*, presented at 1970 NEREM, Boston, Massachusetts, November.
6. Spain, R. J. and H. I. Jauvtis, *High Performance Sequential Memories Using the DOT Technique*, presented at 1971 NEREM, Boston, Massachusetts, November.

Capabilities of the bubble technology

by HSU CHANG

IBM Thomas J. Watson Research Center
Yorktown Heights, New York

INTRODUCTION

This paper aims at a technology forecast for magnetic bubbles. The approach is not to extrapolate the past rate of progress into the future, but to examine the rich varieties of current studies so as to bring attention to what can be evolved to enhance the basic capabilities of the bubble technology.

Superior capabilities are predicted for the magnetic bubble technology—up to 10^9 to 10^{10} bit/in^2 density, less than 1 ms access time even for large files, and versatile switch and logic functions. It should be emphasized that all of these are considered feasible in the context of the laws of physics and desirable in the perspective of competing technologies; but some of these are aspirations rather than accomplishments.

At present, densities of 10^6 and 10^8 bits/in^2 have been realized in packaged memory chips[1,2] and in experimental devices[3] respectively. Bubble memory chips containing up to 20,510 bits per chip were reported by Bell Laboratories,[1] which have been exercised through transfer-read-transfer-circulate cycles in good loops at 100 KHz data rate up to 16 hours. (6.5×10^9 rotating field cycles). Moreover, detailed descriptions have also been given on wafer preparation, uniformity control of epitaxial-film deposition from run to run,[4] circuit element design,[5,6] chip processing,[7,8] and module design.[1] The design and fabrication of a 1024-bit bubble memory chip under test at 100 to 500 KHz were reported by IBM,[2] with accompanying papers on overlay fabrication,[9] drive coil design,[10] packaging considerations,[11] bias magnet configuration,[12] and magnetoresistive sensor characteristics.[13]

There has been substantial progress in the preparation of liquid-phase-epitaxy garnet films, as reflected by reports on the methods of Czochralski growth of high-quality rare-earth gallium-garnets for substrates,[14] the growth of large-area films by liquid phase epitaxy,[15] the effects of growth conditions on film composition,[16] and on the magnetic properties of substituted rare-earth iron-garnets.[17] Film growth kinetics have been the subject of several recent studies[18] where notable progress has been demonstrated toward understanding the liquid-phase-epitaxy process fundamentals. Departing dramatically from the single-crystal approach, sputtered amorphous films of GdCo and GdFe have been reported,[19] which exhibit perpendicular anisotropy and sustain mobile bubbles.

Most of the efforts on improving speed have been directed to increasing mobility in materials, the prevention of hard bubbles and dynamic conversion,[20] and permalloy pattern optimization. Although the ingredients of circuit techniques and memory organizations are available in the literature, their relevance to access and latency times has not been discussed explicitly. As to functional capabilities, many circuit components are described in the patent literature, but critical evaluations and syntheses have been lacking. This paper cannot be claimed as a comprehensive study of all these aspects, but it makes an effort to place them in perspective.

STORAGE DENSITY

The storage density is determined by three factors: bubble size, bubble spacing and lithography capability. The bubble size is primarily determined by the storage-medium characteristics. The bubble spacing is determined by the degree of flux closure provided by the device structure. Moreover, the device structure has associated with it a ratio of overlay line-width to bubble diameter. With a given lithography capability, some device structures can utilize much smaller bubbles (hence higher densities) than the others.

As is well-known, the bubble diameter is given by $d=8l$ in a film of optimum thickness $4l$ (l = characteristic length = $\sigma_w/4\pi M_s^2 = 4(AK)^{1/2}/4\pi M_s^2$, σ_w = wall energy, A = exchange constant, K = uniaxial anisotropy constant, and $4\pi M_s$ = saturation magnetization). Table I lists the storage densities and materials characteristics of several liquid-phase-epitaxy single-crystal garnet films and sputtered amorphous GdCo films. Several industrial laboratories have produced 6 μ bubble diameter materials yielding 10^6 bit/in^2 storage density. Hu et al.[3] have reported operating a 100-bit T-bar shift register at 100 KHz in a material which sustains 0.8 μm diameter bubbles giving a 6.7×10^7 bit/in^2 storage density. The high density is made possible by the high magnetization, $4\pi M_s = 1500$ Gauss, in $(Eu_2Y_1)Fe_5O_{12}$; and also by very narrow width for permalloy bars (4000 Å) prepared by electron-beam lithography. Smaller bubbles have been observed in materials with higher-magnetization materials: for example, 0.43 μm diameter in LPE $(Eu_1Yb_2)Fe_5O_{12}$ films,[21] and 0.08 μm ($Q=H_k/4\pi M_s<1$ at present) to 0.8 μm ($Q>1$) diameter bubbles in amorphous GdCo films.[19] Since the prac-

TABLE I—Storage Densities and Materials Characteristics for Single-Crystal Garnet and Amorphous GdCo Films

MAGNETIC BUBBLE MATERIALS

	THICKNESS (μm)	STRIPWIDTH OR DIAMETER (μm)	CHAR. LENGTH (μm)	$4\pi M_s$ (Gauss)	A (ergs/cm)	K (ergs/cm)	$H_k/4\pi M_s$	DENSITY* (bits/in^2)	COMMENTS
$(Sm_{0.5}Y_{2.5})(Ga_{1.2}Fe_{3.8})O_{12}$	7.4	6.7	.6	192			6.0	10^6	20K-bit chip$^{(1)}$
$(Eu_{.6}Y_{2.4})(Ga_{1.1}Fe_{3.9})O_{12}$		5.0	.55	175	3×10^{-7}	6×10^3	6.0	1.6×10^6	1K-bit chip$^{(2)}$
$(Eu_2Y_1)Fe_5O_{12}$	1.8	0.8	.06	1500	4×10^{-7}	3.5×10^5	3.4	6.7×10^7	100-bit S.R.$^{(3)}$
$(Eu_1Yb_2)Fe_5O_{12}$	0.52	0.43	.045	1600	4×10^{-7}	2.3×10^5	2.27	———	0.4 μm bubble garnet$^{(3)}$
GdCo	1.7 to 3	1.5 to 2.5	.16 to .25	1200 to 2000	$\sim 10 \times 10^7$	$\sim 10 \times 10^5$	1.5 to 4.0	10^7	100-bit S.R.$^{(4)}$
GdCo	1.3 to 2.2	0.8	.039 to .064	$\sim$ 4000			> 1	———	0.8 μm bubble GdCo$^{(4)}$

* T-bar devices ($d = 2w$)		(1)	Bonyhard and Geusic (BTL), Ref. 5.
Bubble dia. $= d$		(2)	Bosch et al. (IBM), Ref. 2.
Bubble spacing $= 4d$		(3)	Plaskett et al. (IBM), Ref. 21.
Bit size $= 16\ d^2 = 64\ w^2$		(4)	Hu et al. (IBM), Ref. 3.

tical limit for electron beam lithography is about 4000 Å at present, the density for bubble devices is not limited by bubble diameter (a materials characteristics), but rather by lithography because the bar width and gap width in T-bar devices must be ½ to ⅓ the bubble diameter.

While the conventional devices (T-bar, Y-bar, X-bar, chevron, etc.) employ narrow magnetic bars to manipulate large bubbles, it is conceivable to have devices employing wide magnetic patterns to manipulate small bubbles. As an example, Wolfe et al.[22] have ion-implanted contiguous disks* into the surface of garnet films to propagate bubbles. In such a configuration, the diameter of the disk patterns is four times the bubble diameter. Wolfe's devices have 10^6 bit/in^2 density with 25 μm diameter disk patterns manipulating 6 μm bubbles. However, if the disk pattern is extrapolated to a diameter of 0.4 μm, the bubble diameter would become 0.1 μm which could yield a density of 4×10^9 bits/in^2. The potential structure density capability exceeds the bubble density capability of the present garnet materials, and matches that of the amorphous materials.

An isolated bubble is an open-flux configuration which seeks flux closure through its surrounding. With permalloy propagation patterns on one side of the storage medium, the flux closure is improved. With ion-implantation, it is conceivable to have magnetic propagation patterns on both sides of the garnet film. Moreover, useful amorphous films seem to require smooth substrates, but do not depend on the substrate material. One may produce device structures with permalloy patterns sandwiching an amorphous film. The two-sided structures should result in steeper field gradient to facilitate propagation and better flux closure to improve density.

In Figure 1, the density vs bubble diameter curves are shown for three different device structures. The upper limits are set by the electron-beam lithography (horizontal bars corresponding to $w = 0.4$ μm). Note that while the conventional T-bar devices offer densities as high as 10^8 bits/in^2, structures such as the contiguous disks could provide densities close to 10^{10} bits/in^2.

For a desired density, the necessary bubble diameter and linewidth for each device structure can be read from the curves. For comparison, both the experimentally achieved and theoretically predicted semiconductor device densities are marked on the righthand side. (Refer to the section on Comparison with Semiconductors).

The quest for high storage density is accompanied by several engineering tradeoffs.

1. Bias field alignment: As high magnetization is used to achieve high density ($d = 8(AK)^{1/2}/\pi M_s^2$), spurious nucleation by the self demagnetizing field must be prevented by high anisotropy field ($H_k/4\pi M_s > 1$) which in turn results in high bias field ($H_{bias} \sim H_k/4$). In the meantime, low-coercivity is maintained to limit

* The devices are certainly very exploratory in nature, particularly in comparison with the well-developed T-bar devices. However, the contiguous disks illustrate how to overcome the lithography limitation. It is also important to develop access, write, and read components consistent with the requirement of manipulating skinny bubbles by fat patterns.

the planar drive field. Thus, a slight misalignment of the bias field would considerably offset the planar rotating field.

2. Planar field amplitude: Larger rotating fields are required for smaller-bubble materials in order to overcome the higher-magnetization bubble-permalloy magnetostatic coupling in jumping gaps.
3. Small signals: Small bubbles result in small signals. Techniques for enhancing the signals by magnetic means will be described in the section on Speed.
4. Fabrication considerations: Although flux closure can be improved by two-sided structures, such structures will add problems in mask alignment.
5. Large capacity at high density: The economy of integrated circuits can be accrued only when high density is accompanied by high storage capacity. As an example, \$100 can buy 10^6 bits at 10^{-2} ¢/bit. At 10^{-3} ¢/bit, it is more reasonable to buy 10^7 bits with \$100, than 10^6 bits at \$10. Large capacity per chip will dictate more functions on the chip so as to limit interconnections and peripheral circuits.

SPEED

The speed of a bubble memory, as measured in terms of data rate, access time, and latency time, can be improved by: (1) storage-medium properties, (2) device structures, (3) electrical propagation of signals, and (4) memory organizations. While the data rate is primarily a function of the storage-medium properties, the access and latency times are significantly influenced by the other factors.

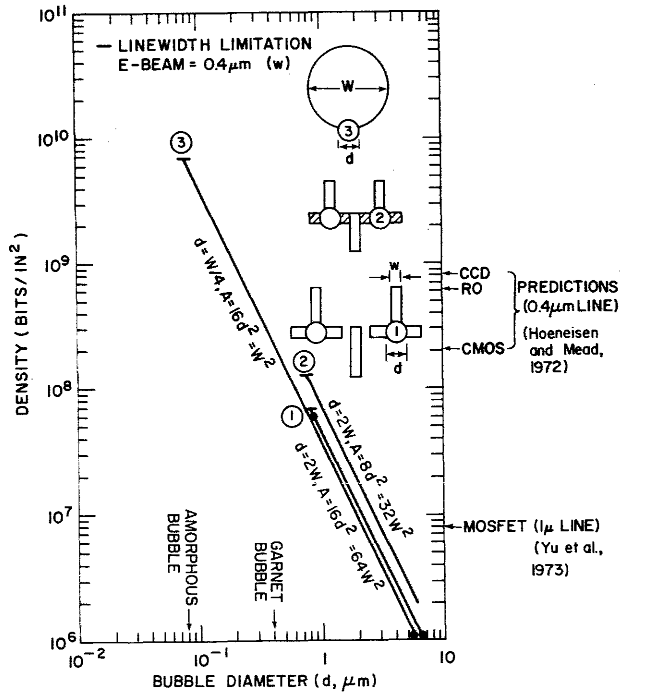


Figure 1—Storage density as a function of bubble diameter, bubble spacing, and lithography capability

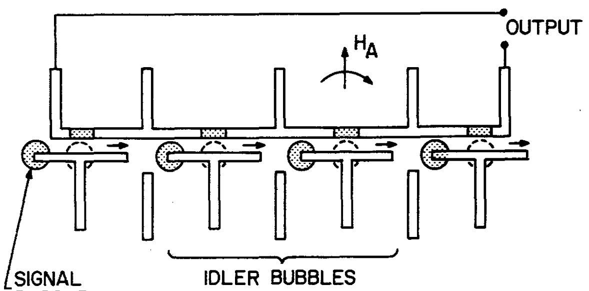


Figure 2—Device structures to amplify signal while maintaining data rate

(1) Storage-medium properties

For optimumly designed propagation structures, the bubble velocity is limited by wall-motion velocity, which is determined by materials characteristics. The bubble velocity is linearly related to the field differential across the bubble at low drive, and limited at a peak value resulting from distortion in wall configuration at high drive. The magnetic parameters, when adjusted to achieve smaller bubbles and higher densities, will also lead to lower mobilities (slopes of velocity vs field curves).

$$\nu = \frac{\mu}{2}\left[\Delta H - \frac{8}{\pi} H_c\right] = \frac{1}{2}\frac{\gamma}{16\alpha}\frac{d}{H_k/4\pi M_s}\left[\Delta H - \frac{8}{\pi} H_c\right]$$

where μ = mobility, ΔH = field differential across a bubble, H_c = coercivity, γ = gyromagnetic ratio, α = damping constant. As a result, the data rate (velocity/bit separation) at low drive remains constant even when the bit size is reduced. The wall-distortion determined peak velocity appears independent of bubble diameter

$$\nu_p = 7.1\ \gamma A/hK^{1/2}$$

where A = exchange constant, h = film thickness, and K = anisotropy constant. Hence the peak-velocity limited data rate (ν_p/4d) will increase with smaller bubbles. Peak velocities on the order of 10^4 cm/sec have been both predicted and observed, corresponding to data rate on the order of 10^7 bits/sec for 2 μ diameter bubbles.

(2A) Device Structures to Enhance Signal

With proper design of magnetoresistive sensors, bubbles of 2 to 6 μm diameters can yield 1 to 3 mV signals.[13] Although this is adequate for the present 10^6 bit/in^2 density (6 μm bubble) memory modules, higher-density devices will require considerable magnetic amplification or more sensitive and expensive sense amplifiers than those in use today.

Archer et al.[23] and Bobeck et al.[6] have described structures which employ more and more chevrons in successive stages to stretch bubbles transversely while they propagate. Since

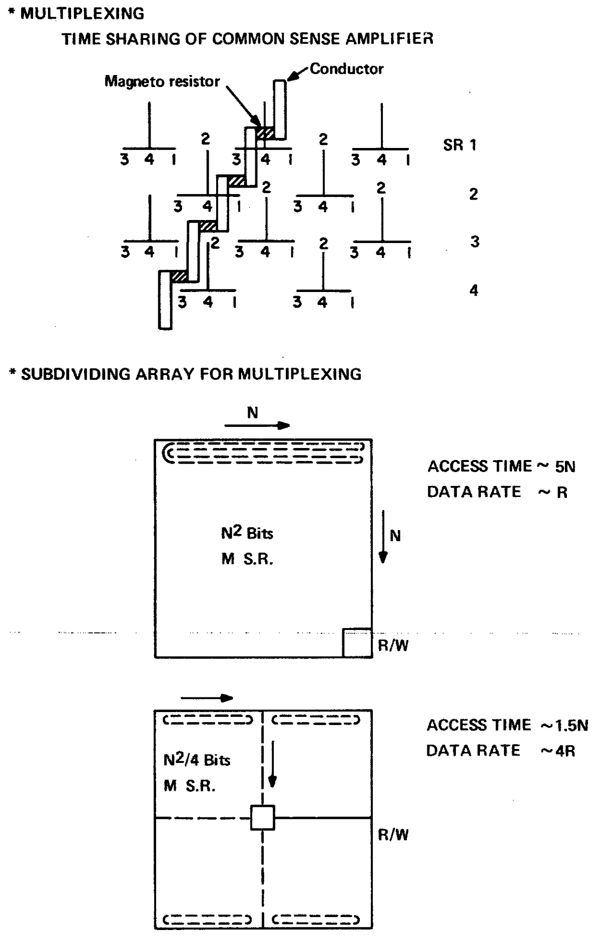


Figure 3—(A) Data rate improvement by time sharing of one sense amplifier by four shift registers (B) Access time improvement by dividing an array for multiplexing

the bubbles are elongated gradually, signal amplification is achieved while maintaining the same data rate. These structures do add to the access time. Additional area is also needed to allow gradual bubble elongation and then contraction before return to the storage loop: (amplification)$^2\times$(area per bit), where the amplification is the ratio of the elongated domain length to the bubble diameter. However, since a sensor is typically shared by 10^4 or more bits, the added sensor area is an insignificant percentage of the chip area.

In principle, the additional access time can be eliminated and the additional area minimized by using a different amplification structure (private communication with Beausoleil and Keefe of IBM). See Figure 2. A compressor is loaded with bubbles, and magnetoresistors are placed adjacent to the idlers and connected in series. As the compressor receives a signal bubble at one end, all idler bubbles are induced to move and produce magnetoresistive signals additively. The signal amplification is accomplished within one field rotation period, and the additional area is merely (amplification)$\times$(area per bit).

Table II—Two Memory Design Examples

DESIGN EXAMPLE 1

16,384 BITS
MAJOR/MINOR LOOPS
128 S.R. EACH WITH 128 BITS

$T_{ACC}=128\ T_R$
$T_{LAT}=128\ T_R$

		PINS	SEMICONDUCTOR CIRCUITS
GENERATOR	(WRITE)	1	1
ANNIHILATOR	(CLEAR)	1	1
TRANSFER	(SELECT)	1	1
SENSOR	(READ)	2	1
GROUND		1	
TOTAL		6	4

DESIGN EXAMPLE 2

268,435,456 BITS
COINCIDENT SELECTION FROM 16×16 ARRAYS
EACH ARRAY: MAJOR/MINOR LOOPS
1024 S.R. EACH WITH 1024 BITS

$T_{ACC}=1024\ T_R$
$T_{ALT}=1024\ T_R$

	PINS	SEMICONDUCTOR CIRCUITS
GENERATOR	16	16
ANNIHILATOR	16	16
TRANSFER	16	16
SENSOR	2×16	16
GROUND	1	
TOTAL	81	64

(2B) Device Structures to Improve Data Rate

For field-access devices, each field rotation period can be divided into four time phases. When signal bubbles from four shift registers are detected by one of four magnetoresistors at four different time phases and fed to one sense amplifier (Figure 3A), the data rate is quadrupled.

In a memory array with N^2 bits, arranged into N/4 shift registers each with 4N bits, (see Figure 3B), the access time is proportional to 5N and the data rate is R. When the array is divided into four quarters with the magnetoresistors located at the center, and each quarter consisting of $N^2/4$ bits arranged into N/4 shift registers each with N bits, the access time is proportional to 1.5N and the data rate is 4R after time multiplexing.

(3) Electrical Propagation of Signals

Bubbles are both the holders and conveyors of information. This dual role facilitates chip organization, data manipulation, and logic operation. However, it is also attended by the disadvantage of bubble propagation delay which is roughly proportional to (bit capacity)$^{1/2}$, and becomes excessive for large arrays. To restrict this delay, the array size must be limited. For a large-capacity chip, it must be organized into a number of smaller arrays each accessible by electrical signals. The task of subdividing a chip into many arrays without increasing significantly the interconnections and peripheral circuits is made possible by memory organization techniques, particularly the coincident selection scheme.

(4) Memory Organizations[24]

Both the major/minor loops and on-chip decoders permit the organization of a large-capacity array into many short shift registers which share interconnections and read and write circuits. The short length as well as the random access capability of the shift registers dramatically reduces the access and latency times. Moreover, both organizations enable the large-area chip to be divided into many smaller arrays which also share interconnections and read and write circuits. Signal transmission by slow bubble propagation is only limited within the small arrays while fast electrical-signal propagation is utilized outside the arrays. Circuit and interconnection sharing is based on the property that only after transfer or decoding operation, can the read or write operation be performed, thus enabling a two-step coincident selection scheme. A further reduction of access time is achieved by the

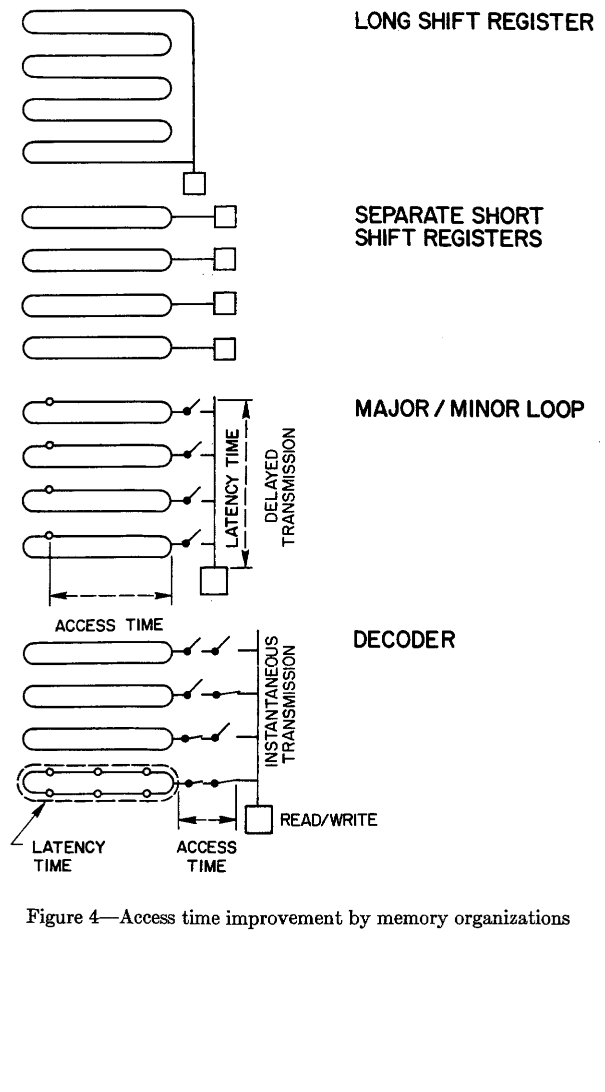


Figure 4—Access time improvement by memory organizations

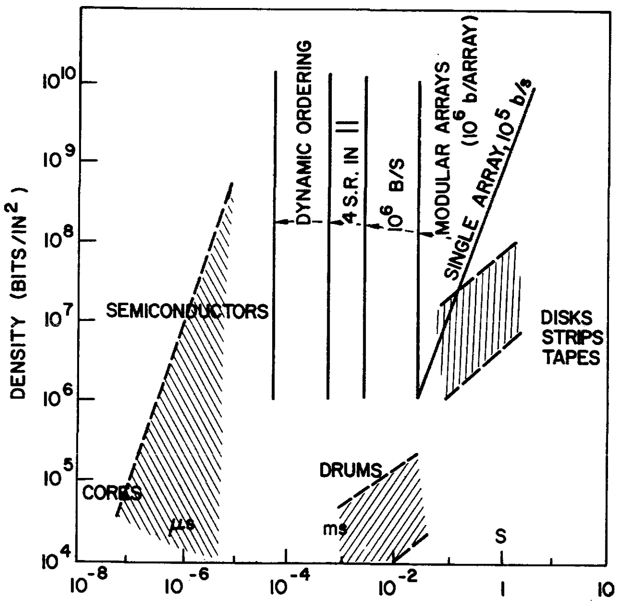


Figure 5—Density vs. access time curves for bubble memory chips

dynamic ordering of data which automatically makes more frequently used data more readily accessible.

DESIGN EXAMPLES

Two design examples are given in Table II to emphasize the salient features of memory organization techniques. In the first example, 16K bits are arranged into 128 minor loops, each with 128 bits and linked by a common major loop. Only 6 interconnection pins and 4 circuits are required for this chip due to the use of the major loop. The maximum access and latency times are both 128 T_R (T_R=field rotation period) due to the short length of the minor loops. In the second example, the chip contains 268 M bits. If it were arranged in a single array of major/minor loops, the interconnections and circuits would remain 6 and 4 respectively but the access and latency times would be 16,384 T_R (e.g., 16 ms for $T_R = 1$ μs), which would not offer speed advantage over a mechanically-accessed magnetic-disk file. Therefore, the chip is divided into smaller arrays arranged into 16 rows and 16 columns. Within each array of 10^6 bits, the bits are organized in major/minor loops, resulting in 1024 T_R for access and latency times. The arrays in each column are connected by a single transfer line while the arrays in a row are connected by single sense, clear, and write lines. The interconnection and circuit counts are respectively 81 and 64. Thus we have increased the capacity by 1.6×10^4 times, with the access time increased only by a factor of 8, and the circuits and interconnections increased by a factor of 16.

The density and speed capabilities will now be summarized together to provide a basis for comparison with other technologies and to assess possible applications for the bubble technology. Refer to Figure 5. Possible densities for bubble devices range from 10^6 to 10^{10} bits/in^2. In order to share the costs of interconnections, circuits and packaging, the chip capacity must be increased with the density.

As a starting point, consider a 1 in^2 area chip with 10^6 bits at 10^6 bit/in^2 density, and 10^5 bit/sec. shift-register data rate. When organized into 10^3 shift registers each with 10^3 bits in a major/minor loop configuration, the access time (maximum) is 20 ms. Now let us increase the chip capacity in proportion to the density, thus maintaining a constant area. A 100-fold increase in density or chip capacity means a 10-fold increase in access time, as indicated by the slanted line labelled "single array". However, the chip can be divided into many 10^6-bit arrays with each array in major/minor loops, and the arrays connected according to the coincident selection scheme. Then within each array the access time is still 20 ms, while the delay incurred from an array to an input/output pin due to electrical signal propagation is negligible, resulting in a net access time of 20 ms regardless of the chip capacity (as indicated by the vertical line labelled modular arrays). When higher data rate (by higher mobility), parallel shift registers, and dynamic ordering are added successively, the access time is further reduced as shown by the various lines in the figure. Note that the benefit of dynamic ordering depends on the nature of data in storage. Only a modest gain of 10 is assumed in the figure.

To allow a comparison of the bubble technology with other technologies, the capabilities of core and semiconductor memories and the mechanical files (drums, disks, strips, and tapes) are also indicated in Figure 5. The description is necessarily sketchy, but the same degree of optimism is exercised for all technologies. Semiconductors and bubbles at the same density and same chip size will offer reasonably comparable cost per bit. However, bubbles will offer lower cost due to simpler processing. Crudely speaking, the cost per bit for the mechanical files will be ten times cheaper than bubbles at the same storage density.

Cores are being displaced by semiconductors in future products due to the increasing density and chip modularity of semiconductors. At densities higher than 10^6 bits/in^2, semiconductors are also going to displace fixed-head-file (FHF) disks and drums when non-volatility is not a critical issue. As for bubbles (a modular non-volatile memory), the application of FHF replacement perhaps provides an opportunity for an early-entry product. The projected very high density suggests that large-capacity files, which at present employ disks, may become a very attractive application. The short access time will eliminate the "file gap" problem.

FUNCTIONAL CAPABILITIES

In the past, magnetic memory devices such as ferrite cores and permalloy films have failed to make any significant inroad in logic applications, while semiconductor devices have extended from logic to memory applications. This is explained simply by the fact that semiconductor devices offer speed as required by arithmetic and logic units, and both memory and logic capabilities to enable integration on the same chip. By contrast, cores and films depend on semiconductors for amplification as is needed in multiple-stage logic operations. They are functionally not self sufficient, and therefore cannot be made into truly integrated-circuit chips.

Bubble devices, although using magnetic materials, have more in common with semiconductors than with cores and films. During each field rotation or current pulsation every bit has its energy replenished by the drive, and its information state re-quantized by the bias field. The bubble memory and logic cells are similar in structure, and therefore amenable to integration on the same chip and even local mixtures.

The bubble logic is not only aesthetically appealing since it makes all-bubble data-processing machines possible;[25] but also practically desirable since it makes integrated-circuit bubble chips realizable. The present memory chips are complete with storage, write, read, and access functions. With larger chips where defects and errors must be more effectively dealt with, switching and logic capabilities are needed for diagnostic testing and redundancy remedy, or error detection and correction. For chips intended for file applications, which have very large capacities (10^8 to 10^{10} bits), data should be arranged and pre-processed before transfer to the memory portion of the storage system.

Should one list a set of desirable attributes for bubble logic, they would include: (1) a universal element, (2) a rewriteable element, (3) multiple inputs, (4) the capability to perform higher level functions (e.g., adder, parity check, etc.) in addition to elemental logic connectives (AND, OR, etc.) and (5) amenability to array logic (or the use of memory arrays of great regularity to perform logic functions of great variety).

These requirements are not fully satisfied by several early bubble logic devices which include conjugate logic gates,[26] chevron 3-3 circuits,[27,28] resident bubble cellular logic,[29] and programmable cellular logic.[30] Continued search has led to new device concepts such as symmetrical switching functions (SSF)[31] which appear to satisfy these requirements. SSF are performed by counting the number of ONE's among the inputs. As an example, consider a three-input SSF element. An AND function will yield an output (ONE) if there are three ONE's in the inputs. An EXCLUSIVE OR function will yield an output if there is only a single ONE among the inputs. For SUM, there is an output if the number of input bubbles is odd. For CARRY, there is an output if the number of input bubbles is two or larger. In canonical forms:

$$\text{AND} \equiv xyz$$
$$\text{EXCLUSIVE OR} \equiv x\bar{y}\bar{z}+\bar{x}y\bar{z}+\bar{x}\bar{y}z$$
$$\text{SUM} \equiv xyz+x\bar{y}\bar{z}+\bar{x}y\bar{z}+\bar{x}\bar{y}z$$
$$\text{CARRY} \equiv xyz+\bar{x}yz+x\bar{y}z+xy\bar{z}$$

A bubble device to perform the SSF can be implemented by executing the following steps: (1) assemble and gravitate input bubbles, (2) convert the number of input bubbles into the position of a bubble in a bubble stream, (3) personalize a bubble stream to define the logic function, and (4) interact the two bubble streams to produce the output of the logic function.

Note that if simple logic connectives are used to construct the SUM and CARRY devices, NEGATION devices are needed to generate $\bar{x}$, $\bar{y}$, and $\bar{z}$; AND gates (one three-input element or three two-input elements arranged in two stages) are needed to generate xyz, $x\bar{y}\bar{z}$, etc.; and OR gates are needed to produce sum from xyz, $x\bar{y}\bar{z}$, etc. Thus a variety of devices will be needed to execute elemental functions in stages, with demand of precise path length for precise timing.

COMPARISON WITH SEMICONDUCTORS

Given that both bubbles and semiconductors are integrated-circuit technologies, the relative complexity in device fabrication can be assessed in terms of the complexity in planar configurations (particularly in the line width w) and the number of processing steps (particularly the number of masking steps). One convenient measure of the efficiency of device design is to express the area of a cell in units of w^2:

	Cell Area	Present Density (bits/in^2)	Potential Density (bits/in^2)
MOSFET	30-50 w^2	8×10^6 (Ref. 32)	1.3×10^8
T-bar bubble devices	64 w^2	6.7×10^7 (Ref. 3)	6.7×10^7
Contiguous-disk bubble devices	w^2	10^6 (Ref. 22)	4×10^9

The potential densities are based on the practical limit of electron beam lithographies, $w = 0.4$ μm. In integrated structures, the yield of the complete structure results from the product of the yields of the successive steps. Obviously the fewer the processing steps are, the higher the overall yield becomes. In MOSFET typically five masking steps are needed. By contrast, the conventional T-bar devices require three photoresist masking steps, and recently Bobeck et al.[6] have evolved a one-mask overlay design. In addition, it should be observed that the structures for bubble devices are merely overlays, while the structures for semiconductor devices are integrated into the Si medium. One may conclude that bubble devices are intrinsically and significantly simpler in structure and fabrication as compared to the MOSFET devices.

The potential density for semiconductor devices is consistent with the prediction by Hoeneisen and Mead.[33] They considered the physical phenomena which will ultimately limit circuit miniaturization of MOS field-effect planar transistors with Si substrate and SiO_2 dielectric, and concluded that the potential densities for dynamic or complementary MOS, read-only memory cells, and charge-coupled devices are respectively 2×10^8, 6×10^8 and 8×10^8 devices/in^2.

COMPARISON WITH MAGNETIC DISKS

The magnetic disks have offered large-capacity on-line read/write storage with attractive cost and reasonable speed. For fast-access fixed head disk files, the capacity is in the range of 10^7 to 10^8 bits, and the access time is in the range of 5-50 ms. For large-capacity movable-head disk files, the capacity is in the range of 10^9 to 10^{10} bits, and the access time is in the range of 20 to 300 ms. From its very inception, the bubble technology has been heralded as a solid-state device to replace disk files. Note that the state-of-the-art bubble-memory modules functionally and cost-wise can only be compared with fixed-head disk files. However, the projected high-density bubble devices may indeed impact large-capacity files.

Disk files have become the dominant large-capacity storage[34] due to its low cost, which is achieved through the large number of bits in the inexpensive storage medium (e.g., 10^9 bits on 12 disks in the IBM 3330 disk file) sharing the expensive mechanical drive, servomechanisms, channel electronics, control units, etc. The large number of bits on disks mounted on the same spindle is made possible by high storage density. As a disk is rotated, the bits along a circumferential track are made accessible to the read and write heads. Three geometrical dimensions are crucial in determining the linear recording density: the gap length of the head, the head-to-medium separation, and the medium thickness. As a rule of thumb, the three dimensions are comparable, and their sum is the linear bit dimension. For example, at 20,000 bit/in linear density, the linear bit dimension is 5×10^{-5} in (50 μin, 1.27 μm, or 12,700 Å), and approximately the gap, the separation, and the thickness are all only 4,230 Å. Thus thin-film heads and disks become candidates to fulfill the exacting dimensional requirements; and smooth surface, continuous air flow, and clean ambient become essential to ensure a constant separation. To achieve such technical objectives is no small task, which encourages more tolerant thoughts toward exploring competing new technologies.

To contrast the disk storage medium and the bubble storage medium, the former needs only to offer its intrinsic storage capability and relegates the access, read and write functions to other components in a disk file; while the latter must have device structures built into the chip to provide all of the storage, access, read and write functions. Naturally, the latter would be more expensive to fabricate, on a per-unit-area basis. In order to compete with magnetic disks on a cost-per-bit basis, bubbles must attain higher storage density (ten times or more).

In comparison with solid-state devices, mechanically-accessed magnetic disks offer satisfactorily high data rate, but unsatisfactorily slow access time. At 14″ disk diameter, 20,000 bits/in, and 3000 rpm, the data rate is 42×10^6 bits/sec. The rotation period or the latency time is 20 ms. To select a track by moving the arm carrying the slider and head, the access time ranges from tens to hundreds of ms. Bubble devices do not offer high data rate; 0.1 MHz in practical devices now, and potentially 10 MHz in future devices. However, the ability to switch data electronically from one shift register to another permits great flexibility in memory organization to achieve real or effective access time as low as tens to hundreds of microseconds.

If and when bubbles become competitive with disks in terms of cost per bit, then all of the system applications which require non-volatility, rewriteability, and on-line and off-line storage, can employ bubble files rather than disk files. The absence of mechanical motion will make bubble files more reliable. Since bubbles do not critically depend on extremely large module capacity for low cost per bit, bubble memories and storages can offer a wider range of capacities. Through memory organization, much faster access time is obtainable with bubble files. Moreover, the various parts in a bubble file can operate synchronously or asynchronously. Bubble files can have their own bubble memories, perhaps even integrated onto the same chips, in contrast to disk files employing core memories with complicated channel electronics in between. The switching and logic capabilities can be used for data arrangement as well as pre-processing before transfer to a memory.

CONCLUSIONS

Packaged bubble memory chips offer 10^6 bits/in^2 density. Experimental shift registers approach 10^8 bit/in^2 density. Moreover, simultaneous improvement in materials, device structure, and lithography could lead to storage density as high as 10^9 to 10^{10} bits/in^2. Both in terms of current experimental devices and theoretical predictions, bubbles offer ten times higher density than semiconductors. In addition, bubbles may use only one masking step while FET devices require five.

Packaged bubble memory chips operate at 10^5 bit/sec data rate or faster. Improved materials are expected to yield 10^7 bit/sec data rate. Memory organization techniques such as on-chip decoders or major/minor loops, when aided by dynamic ordering or coincident selection could lead to access time order-of-magnitude shorter than the 10 ms for the fast disk files.

Semiconductors offer great functional versatility to be a self-sufficient technology; high speed to yield tremendous processing power; and integrated-circuit fabrication to permit low-cost, reliable, and convenient packaging.

Bubbles do have memory, switching, and logic capabilities. Therefore, bubbles far surpass any other magnetic technology in approaching the functional versatility of semiconductors. Bubble logic devices so far have received little attention from the experimentalists. Nevertheless, the possibility of universal, re-writeable, multiple-input devices suitable for higher-level functions and array logic is quite obvious. Thus bubbles are potentially capable of reaping the full advantage of integrated circuits such as enjoyed by semiconductors, including density and speed, functional versatility, batch fabrication, and convenient package.

Magnetic disks offer non-volatility, re-writeability, and on-line or off-line storage. So do bubbles. The very low cost of disks makes very large capacity affordable (say 10^{10} bits). Actually, the other way is also true; viz. the large capacity makes low cost possible, by amortizing expensive peripheral components against many bits Magnetic bubbles require more expensive materials and complicated structures than disk devices Hence bubbles will impact disk files only when a density advantage of a factor of 10 or higher is achieved However, bubbles are modular, fast (millisecond or sub-millisecond access time), and offer large data bandwidth by connecting chips in parallel As the bubble technology matures to impact the disk files, it may well change today's memory hierarchy because it can and therefore it will incorporate memory, data manipulation, and data processing on the same chip.

ACKNOWLEDGMENTS

The author gratefully acknowledges the many stimulating discussions with his colleagues: C. D. Cullum, J. H. Eaton, G. C. Feth, H. L. Hu, G. E. Keefe, Y. S. Lin, L. L. Rosier and J. C. Slonczewski.

REFERENCES

1. Michaelis, P. and P. I. Bonyhard, *IEEE Trans. on Mag.*, 9, No. 3, pp. 436-39, September 1973.
2. Bosch, L. J., et al., *IEEE Trans. on Mag.*, 9, No. 3, pp. 481-3, September 1973.
3. Hu, H. L., et al., *1973 Intermag. Conf. Digests*, Paper 26.5.
4. Hewitt, B., et al., *IEEE Trans. on Mag.*, 9, No. 3, pp. 366-72, September 1973.
5. Bonyhard, P. I., et al., *IEEE Trans. on Mag.*, 9, No. 3, pp. 433-5, September 1973.
6. Bobeck, A. H. et al., *IEEE Trans. on Mag.*, 9, No. 3, pp. 474-80, September 1973.
7. Reekstin, J. P. et al., to appear in *J. Vac. Science and Technology*, 1973.
8. Johnson, W. A. et al., *1973 Intermag. Conf. Digests*, paper 26.3.
9. Powers, J. V. and R. E. Horstman, *1973 Intermag. Conf. Digests*, paper 21.1.
10. Kiseda, J. R., *IEEE Trans. on Mag.*, 9, No. 3, pp. 425-8, September 1973.
11. Rifkin, A. A., *IEEE Trans. on Mag.*, 9, No. 3, pp. 429-32, September 1973.
12. Lyons, W. A., *1973 Intermag. Conf. Digests*, paper 26.6.
13. Almasi, G. S., *Proc. IEEE*, Vol. 61, No. 4, pp. 438-444, April 1972.
14. Brandle, C. D. and A. J. Valentino, *J. Cryst. Growth*, Vol. 12, pp. 3-8, 1972.
15. Blank, S. L., et al., *AIP Conf. Proc.*, No. 10, Pt. 1, pp. 256-270, 1973.
16. Giess, E. A., et al., *J. Crystl. Growth*, Vol. 16, pp. 36-42, 1972.
17. Giess, E. A., et al., *AIP Conf. Proc.*, No. 5, Pt. I, pp. 110-114, 1972.
18. Ghez, R. and E. A. Giess, *Mat. Res. Bull.*, Vol. 8, No. 1, pp. 31-42, 1973.
19. Chaudhari, P., et al., *IBM J. Res. and Dev.*, Vol. 17, No. 1, pp. 66-68, January 1973.
20. Vella-Coleiro, G. P., *AIP Conf. Proc.*, No. 10, Pt. I, (18th Conf. on Magnetism and Magnetic Materials), pp. 424-441, 1973.
21. Plaskett, T. S., et al., *AIP Conf. Proc.*, No. 10, Pt. I (18th Conf. on Magnetism and Magnetic Materials), pp. 319-323, 1973.
22. Wolfe, R., et al., *AIP Conf. Proc.*, No. 10, Pt. I, (18th Conf. on Magnetism and Magnetic Materials), pp. 339-341, 1973.

23. Archer, J. L., et al., *IEEE Trans. Magnetics,* Vol. MAG-8, No. 3, pp. 695-700, September 1972.
24. Chang, H., *IEEE Trans. Magnetics,* Vol. MAG-8, No. 3, pp. 564-568, September 1972.
25. Minnick, R. C., et al., *Proc. 1972 Fall Joint Computer Conf.,* pp. 1279-1298, 1972.
26. Sandfort, R. M. and E. R. Burke, *IEEE Trans. Magnetics,* Vol. MAG-7, No. 3, pp. 358-360, September 1971.
27. Bobeck, A. H. and H. E. D. Scovil, *Scientific American,* Vol. 224, No. 6, pp. 78-90, June 1971.
28. Minnick, R. C., et al., *WESCON Proc.,* 8/4, pp. 1-13, 1972.
29. Garey, M. R., *IEEE Trans. Computers,* Vol. C-21, No. 4, pp. 392-396, April 1972.
30. Bobeck, A. H. et al., *U.S. Patent 3,* 541,522, filed 8/2/67, issued 11/17/70.
31. Chang, H., et al., *AFIPS Conf. Proc.,* Vol. 42, pp. 413-420, 1973.
32. Yu, H. N., et al., *1973 ISSCC Digest of Technical Papers,* THAM 8.6, pp. 98-99.
33. Hoeneisen, B. and C. A. Mead, *Solid State Electronics,* Vol. 15, pp. 819 and 891, 1972.
34. Harker, J. M. and H. Chang, *Proc. Spring Joint Computer Conf.,* pp. 945-55, 1972.

Some computer network interconnection issues*

by A. M. McKENZIE

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

During the past four years, my work at Bolt Beranek and Newman has been devoted to the operation and use of the ARPA Communication Network.[1] Further, my only direct experience with computer communication networks is this involvement. It would therefore be brash to present myself as an expert on digital communications, and I do not presume to do so. On the other hand, the ARPA experiment of constructing a communications system specifically tailored to the needs of computers and their users has been extremely successful, and at the same time has pointed out a number of areas where methods of computer communication can stand improvement. It is this background of experience which has inspired the following observations.

I have no doubt that the problems of interconnecting widely dissimilar computer networks can be rapidly overcome as soon as there is economic or intellectual stimulus to make the interconnections. The most probable method, I suspect, is to insert a small computer between two such networks, so that it appears as a customer to both, and performs elementary transformations of data which must cross from one to the other. Examples of such a gateway device come easily to mind: the University of London would like to operate a gateway between the ARPA, NPL, and CYCLADES networks; for a few days in 1972 an ARPA Network Terminal IMP (TIP) served as a gateway between the ARPA Network and TYMNET; a TIP or an ANTS can be thought of as a gateway between the ARPA Network and the dial network provided by the common carriers.

Since this gateway approach works adequately (if not elegantly), I am not bothered by a lack of standards for network interconnection. In fact, given the youth of the "packet-switching" network field, and the high probability that there is still much to be learned, I am greatly opposed to a move to standardize on some particular network design in the near future.

In spite of what I have just said, there are real problems for potential users of a data network or a series of interconnected networks. Interconnection of a series of private and public mail systems can and does move a letter from Paris to Chicago, but that does not do the recipient much good if the letter is written in French and he reads only English. It is much the same with a network (or set of networks) of heterogeneous computers: what should a DEC PDP-11 do with an IBM 370 floating point number? I suppose that one can imagine another set of gateways, let us call this set "translators," through which data is routed and which transform from one "language" to another. (In the ARPA Network such a service, the "Data Reconfiguration Service",[2] was proposed, but thus far it has proved more practical to invent a common language and for all to learn to speak it; "Esperanto" appears practical with computers where it is not with humans.)

But there is an even more difficult set of problems in computer networks, a set which does not lend itself so well to human analogy. This set revolves around buffering and synchronization between asynchronous processes. There has been a great deal of discussion in the literature about the coordination of cooperating processes within a single computer system, notably the P and V semaphores of Dijkstra.[3] Such systems, however, tend to assume (1) an omnipotent supervisor, and (2) essentially perfect communication channels with zero delay. Because of these assumptions, I am inclined to believe that these coordination schemes do not extend well to computer networking.

Let me discuss, then, the issues which I believe are fundamental to communication among dissimilar computers ("Hosts") connected together through a series of dissimilar communications networks ("nets"). I will refer to the rules which govern the communication as "protocols"; these protocols must be designed to provide solutions to the buffering and synchronization problems mentioned above.

ONE PROTOCOL OR MANY

I would suggest that the development of a standard set of protocols should not close off all the options for private arrangements. There should be easy methods for pairs (or larger groups) of customers to escape to something different. This is partly because I consider it unlikely that the first round (or first "n" rounds) of standards development will make all the right choices, and partly because sets of identical computers will probably find a specialized protocol of their own more efficient than any general-purpose standard.

* This work was sponsored by the Advanced Research Projects Agency under Contracts DAHC15-69-C-0179 and F 08606-73-C-0027.

THROUGHPUT VS. DELAY

It has generally proved true in the past that different mechanisms are required for maximizing throughput and for minimizing delay. Computer traffic typically has both types of requirements; e.g., file transfer may have high throughput requirements while interactive users, or one system calling a subroutine on another system, are low bandwidth but low delay applications. Thus it is reasonable to speculate that standard protocols should provide different mechanisms to deal efficaciously with these very different types of traffic.

"CIRCUITS" OR "LETTERS"

The fact that a net provides service by packet switching (or circuit switching) has thus far had only minor influence on whether the Host protocols have followed a similar philosophy. For example, the ARPA Network is a packet-switching communication service but the Host protocols are currently based on "logical circuits"; Walden[4] has proposed an interprocess message protocol which appears to generalize well to network or multi-network operation, but it has not been tested experimentally. On the other hand, multidrop polling systems tend to use a message-oriented protocol in a circuit-oriented communications environment. It may be that the choice is related to the throughput/delay issue mentioned above; perhaps a circuit protocol is most suitable for large volume, high throughput use while a message protocol is most suitable for low volume, low delay applications. In this regard, the comparison of packet- and circuit-switching networks by Itoh and Kato[5] may be relevant.

RETRANSMISSION AND ACKNOWLEDGMENT

Any communication system has the possibility of occasionally losing data. Circuits suffer from burst errors, complex switches are susceptible to occasional failure. Thus it seems obvious that protocols for any communication system must include methods for erroneous or lost message detection and correction. A message protocol might best be served by sequential numbering of messages, positive acknowledgment by the receiver, and timeout and retransmission by the sender. (Sender and receiver could, of course, agree to ignore lost or erroneous messages for some applications.) For circuit protocols, especially over full-duplex circuits, Pouzin[6] has argued convincingly that the most efficient acknowledgment procedure is based on the subdivision of a channel (logical or physical) into several independent subchannels, each with its own independent acknowledgments. One chooses the number of subchannels according to the speeds of the sender and receiver and according to the expected network delay. This scheme is in contra-distinction to the ISO proposal for a High-Level Data Link Control[7] and to the proposal for experimental "Protocols for Internetwork Communication" of Cerf and Kahn.[8]

NETWORK TIMING

Circuit-switching networks tend to impose some circuit setup delay at the beginning of a communication, but only a fixed and well-known propagation delay during the communication. Packet- and message-switching networks, on the other hand, impose arbitrary delays due to queueing and for other reasons on each element of a communication although there may be little or no setup time. In my opinion it makes little sense for two Hosts, each connected to a network of low delay, to attempt to communicate through an intermediate net of very high delay; this becomes obvious when one tries to pick timeout/retransmission parameters. Thus a real issue in the design of standard protocols is the set of nets one expects to work through; very different strategies should be followed depending on whether the expected delays are measured in minutes (or hours or days as in conventional torn tape systems) or in tenths of a second.

SIMPLICITY AND TABLE SPACE

I believe it should be possible, once internetwork protocols are established, to connect some customer equipment to some of the nets via simple hardware devices. I envision, for example, a hardware multiplexor which services dozens (or hundreds) of interactive terminals and which handles all of the necessary protocol. Thus I am not greatly enamored of protocols which require large volumes of tabular data, which require garbage collection of tabled data, or elaborate buffer management strategies to service interactive terminals. Similarly, I can imagine trying to connect a magnetic tape drive more or less directly to a net, again with a fairly simple interface. Here I may be willing to pay (in complexity) for high throughput but not be very interested in delay. The point is that there should probably be several different protocols, each no more complex than is necessary for a particular class of application. After all, in a computer communication network it is reasonable to obtain considerable computational power from remote systems; the standard protocols may only need to be sufficient to reach these resources. A similar approach, although at a different level, has been used quite successfully in the ARPA Network's Terminal IMPs.[9]

COMMUNICATION EFFICIENCY VS. HOST PROCESSING

Traditionally, one of the primary concerns in data communication has been to squeeze every possible bit out of the information being carried by a communication circuit. It is argued that with powerful communications computers at each end of the circuit, the cost of processing required to remove and regenerate bit patterns is insignificant compared to the cost of the circuits and the delay incurred by long messages. This argument begins to lose force, however, when the computers at each end of the circuit are not primarily concerned with communications but with other tasks, and as the cost of communicating is reduced. In a network of heterogeneous computers it is probably more efficient in the long run to pad out messages, or blocks, so that the meaningful data is easily manipulated by both the sender and receiver. In the ARPA Network, for example, the protocols call for padding the header of each message to 72 bits, the least common multiple of 8, 18, and 36, so that bit-shifting is reduced to a minimum in the concatenation of several messages making up a single file, regardless of the word lengths of the source and destination computers.

FRAGMENTATION AT NETWORK BOUNDARIES

When considering the interconnection of several nets, each of which may have some maximum permitted block or message size, one is forced either to permit fragmentation at the boundaries or to adopt, as an internetwork standard, a maximum block length which is no greater than the minimum for all the nets. Assuming that the minimum is reasonable (say 1000 bits or more), I am inclined to favor a mixed approach based on the throughput/delay requirements. Constraining low volume, low delay messages to fit within a single block, while permitting fragmentation of high volume traffic, seems a reasonable choice. Cerf and Kahn[8] present one scheme for dealing with fragmentation problems, although, as previously indicated, I would prefer the separation of the reordering and the acknowledgment mechanisms.

ACKNOWLEDGMENT

My ideas on standard protocols and other network interconnection issues are the result of continued interaction with colleagues physically located both at Bolt Beranek and Newman and at other ARPA Network sites. I am particularly indebted, however, to David C. Walden of BBN.

REFERENCES

1. Roberts, L. G. and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *AFIPS Proceedings*, SJCC, Vol. 36, 1970.
2. *Status Report on Proposed Data Reconfiguration Service*, ARPA Network Working Group RFC #138, NIC #6715, April 1971.
3. Dijkstra, E. W., "Cooperating Sequential Processes," *Programming Languages* (Genuys, F., ed.), Academic Press, N.Y., 1968.
4. Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," *Communications of the ACM*, Vol. 15, No. 4, April 1972.
5. Itoh, K., and T. Kato, "An Analysis of Traffic Handling Capacity of Packet Switched and Circuit Switched Networks," *Proceedings of the Third Data Communications Symposium of the IEEE Computer Society and the ACM*, 1973.
6. Pouzin, L., *Efficiency of Full-Duplex Synchronous Data Link Procedures*, International Network Working Group Note #35, NIC #18255, June 1973.
7. *Proposed Draft International Standard on Commands and Responses*, ISO/TC 97/SC 6 (Geneva-4) 732, February 1973.
8. Cerf, V., and Kahn, R., *HOST and PROCESS Level Protocols for Internetwork Communication*, International Network Working Group Note #39, NIC #18764, September 1973.
9. Mimno, et al., "Terminal Access to the ARPA Network: Experience and Improvements," *COMPCON 73, Seventh Annual IEEE Computer Society International Conference*, 1973.

Step-by-step—A career structure for systematic EDP growth

by NANCY L. AYER

National Agricultural Library
Beltsville, Maryland

The mushrooming growth of data processing applications in the past decade has been a boon in many ways for the country. One of the problems, however, has been that growth was so rapid and so demanding that there was little time or manpower resources left over in which plans and standards or other procedures could be formalized. Only in the recent past has there been time to go into this area. Some fragmented standards have appeared here and there and it is now time to draw these together and provide one standard guideline to help managers, professional EDP personnel, as well as those desiring to enter the EDP career area.

In recognition of the need to fill this void, last year a small group in the Washington, D. C., area got together to approach this very important matter. Paul Oyer established this under the shelter of a special interest group of ACM and he was joined by myself plus many other idealistic pragmatists such as Dave Skeen, Bob Meyer, Mary Jo Blair, Ken King. Our purpose was to provide a set of documentation to serve as a fairly comprehensive guideline for the industry. We recognize in advance that it will not answer the needs of everyone or be a be-all-to-end-all. On the other hand, as everyone knows, it is easier to take documentation already provided and change it than it is to create it in the first place. This project, then, is to provide one set of guidelines which would be presented to a Board of Review to be selected from experienced persons of integrity and vision from industry, government, managers, supervisors, professionals in neither the managerial nor supervisory areas, professional associations, and academic institutions. The approved document will then become a recommendation for use by individuals and organizations in management planning and individual career development planning.

THE PLAN

The plan as presently proposed is a five-pointer. *Point 1*: Establish a career path network for data processing personnel. Extensive work done previously by a consulting firm reduced over 1700 positions to a basic 54 positions. In the group selected, these 54 general position descriptions included 189 variations by the addition of locally desirable requirements. This is, in fact, the entire thrust of the project: namely, to provide a basic set of guides which can be customized to individual organization needs. The positions resolved themselves into ten (10) family clusters.

- *EDP Management*—management of branches, divisions, and groups of divisions.
- *EDP Research and Development.*
- *Technical Supervisors*—support and first-line supervisors.
- *Systems Specialists*—major systems development, design, and data base development.
- *Programming*—writing and maintaining computer programs.
- *Technical Specialists*—computer operations analysis, software at the systems level, and applications system analysis and design (although this latter is still a matter of controversy).
- *Production Management*—scheduling, directing, controlling, and coordinating the operation of computer complexes.
- *Equipment Operations*—operation of all computer complexes and peripheral hardware.
- *Technical Support*—collection, assembling and delivery of materials among functional areas of EDP and provision of clerical support in these areas.
- *General Support*—EDP training and administrative support.

Following the identification of these positions within family clusters, a relative level is assigned to each. The levels presently utilized are those of the governmental GS (General Schedule) standards.

Point 2: Establish basic descriptions of each position including:

- *Knowledge required* by the job. General and specific tasks.
- *Responsibilities.* General statement of commitments, supervisory control; and impact of work.
- *Difficulty of the level.* Nature, complexity, scope, guidelines provided, judgment.
- *Personal Relationships.* Contacts and purpose of such contacts.
- *Environmental Demands.* Variables depending on hardware, software systems, but identifiable as pertaining to a specific level.

- *Performance Measurement.* Specific achievement requirements.

Point 3: For each level of position, develop achievement points which must be available to meet the position requirements, and to pinpoint those which may be achieved by academic means as a substitute for experience. Total academic credits may not be substituted, however. Some of these attributes are:

- *Qualifications* of the position itself.
- *Employee appraisal* on such matters as quantity of work and quality, analytical reasoning, resolution of problems, initiative, oral communication, etc.
- *Test scoring* if a prerequisite.
- *Experience.* An example of this would be experience indicating skill in planning and developing machine logic, and program steps for preparing instructions for machine processing. If a total of 14 points were given to this item, a substitution of 7 points could be given for each full year of experience in a position where only two years experience were required. No excess over the 14 would be allocated.
- *Training and self development.* This could include a graduate college degree if major areas of study have provided skills and knowledge relative to the position; undergraduate college degree with the same provision; or partial credit for courses in the field being considered.
- *Awards* received.
- *Outside activities* which indicates SIGNIFICANT participation in matters proving aptitudes and ability for the position.
- *Encourage certification examinations* for each level of each position family.

Point 4: Provide logical crossover channels between family clusters, and from position to position within a family. This is, of course, the point most desired by the EDP professional or would-be professional for his own advancement. An overlooked advantage, however, would also be for the manager or supervisor in counselling and planning upward mobility for his personnel. It is of immense importance to the vast majority of workers to have an upward goal in mind. It is of even more importance for each employee to know just how he must proceed in order to reach the goal effectively. In developing logical crossover points and channels, a reasonable road map is established together with the means of arriving at a predetermined destination. Note on the Exhibit 5 (sample crossover chart from the computer operator cluster to the programming cluster) that no transfer is permitted from entry level operator. It has been determined that sufficient responsibility must be exercised by the individual to complete one phase of his training before attempting to transfer. Therefore, the operator must complete entry level and be in the process of working as a non-entry level employee in that field before a crossover is deemed desirable. All positions for which criteria are to be established must of necessity assume that certain basic prior knowledge is inherent in experience at a certain level. Thus, although the experience of a programmer II specifies certain knowledge of EDP terminology, other knowledge obtained at a lower level of experience must be assumed. For this reason, the programmer II experience would not specifically designate basic hardware knowledge and thus must be obtained either as a programmer III or computer operator II or III.

Note the crossover chart also specifies certain courses which may be substituted for some experience. Exhibit 6 shows a sample listing of course names, and Exhibit 7 contains a sample course description. The initial character specifies the family—in this case the P stands for the programmer family. The suffix indicates the level of instruction required:

0 =	orientation
1 =	basic course
2 =	journeyman level
3 =	advanced material
4 =	specialized material

One additional point (Point 5) which is dealt with more thoroughly by Bob Meyer's Group would be a compilation of courses within a geographic area so that each individual could know where to find the course required.

CONCLUSION

This then is the plan. To provide a road map by means of which everyone concerned would know where he stood in the scheme of things. It would no longer be necessary for an employer to guess from a resume whether or not an applicant was qualified. It would no longer be possible for an employee not to know his necessary experience or academic requirements for the position of his choice. By also inserting examinations (not the same thing as certifications) for each level of position, everyone concerned would know if a professional or his would-be counterpart were qualified for a specific position level.

Such a procedure would give dignity to a profession. It would tend to give integrity to the professional. It would also give a sense of purpose and measurement for a methodical progression.

It is realized that such a set of guidelines would have to be updated periodically, module by module. But is not this the basis of a progressive profession? To develop such a set of guidelines is not an easy matter. We have been fortunate to have such a good start on it, and from the interest generated at all levels for the completed project (or any part of an interim product), there should be many interested in helping to carry it on. As a matter of fact, Bob Henry of the University of Minnesota, has already volunteered his resources and the Federal ADP Users Group of Washington, D. C., has asked us to act as a special interest group under their auspices. What a really great helping hand these people are.

Only by having the backing of all aspects of the field will it be successful—academic community in planning and providing curriculum; professional societies in encouraging acceptance, testing, certification, and professionalism; managers in using it as a guide in planning and operating their establishments; and last but far from least, the individual himself.

SELECTED BIBLIOGRAPHY

1. Canning, Richard G., "The Upgrading of Data Processing Personnel," *EDP Analyzer*, August 1967.
2. Caruth, Donald L., *Guidelines for Organizing a Work Measurement Program*, Association for Systems Management, Cleveland, Ohio, 1971.
3. *Data Processing Career Structures and Development*, Volumes I and II. Brandon Applied Systems, Inc., Arlington, Va., 1972.
4. DeMaagd, Gerald, "Matrix Management" *Datamation*, Oct. 15, 1970.
5. Dickman, Robert A., *Personnel Implications for Business Data Processing*, Wiley-Interscience, 1971.
6. Greco, Ben, *The Importance of Career Planning Prior to Job Search*, MBA, December 1970.
7. Henry, Robert M., "Skills Possessed and Skills Useful for MIS Practioners: A Research Report," *PROC of NCC '74*, Volume 43.
8. Kaye, Donald, "Career Paths in Systems and Data Processing," *Journal of Systems Management*, June 1971.
9. Oyer, Paul D., Dorothy L. Ray, "A Systems Approach to Career Development: Report of Two Surveys," *PROC of '74 NCC*, Volume 43.
10. Pigors, Paul, Charles Meyers and F. T. Malm (eds.), *Management of Human Resources*, McGraw-Hill, New York, 1969.
11. Rigney, Joseph W., R. M. Berger and A. Gershon, *Computer Personnel Selection and Criterion Development: III. The Computer Position Profile*," University of Southern California, Los Angeles, 1967.
12. Skeen, David R., "EDP Certification . . . Is It Necessary?" *PROC of '74 NCC*, Volume 43.
13. *The Systems Approach to Training Development: The Career Path Method*, Development Systems International Corporation, Studio City, California, 1971.
14. Wolfe, Jack M., "Testing for Programming Aptitude," *Datamation*, April 1969.

Exhibit 1

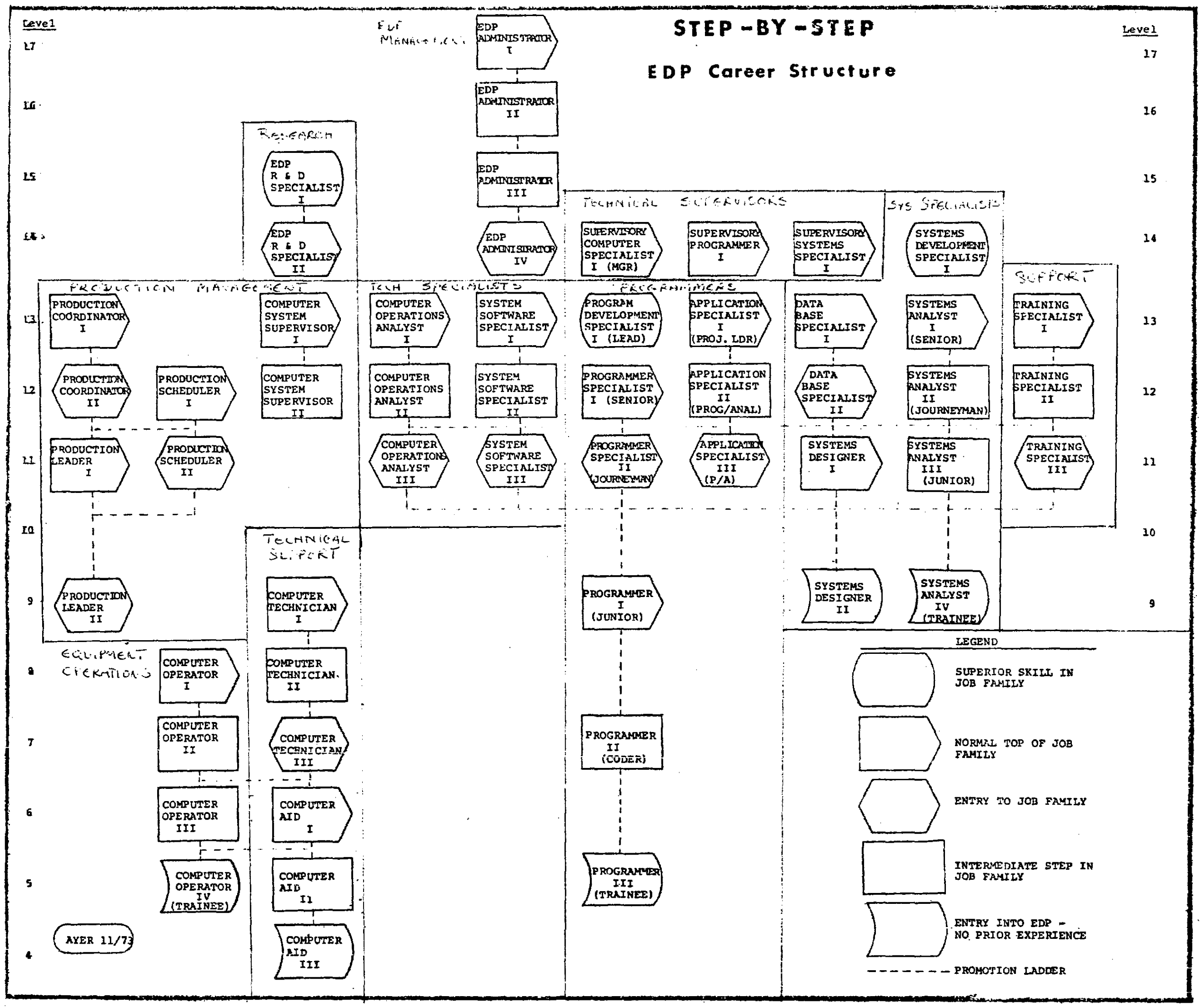


Exhibit 1

Exhibit 2

EQUIPMENT OPERATIONS FAMILY

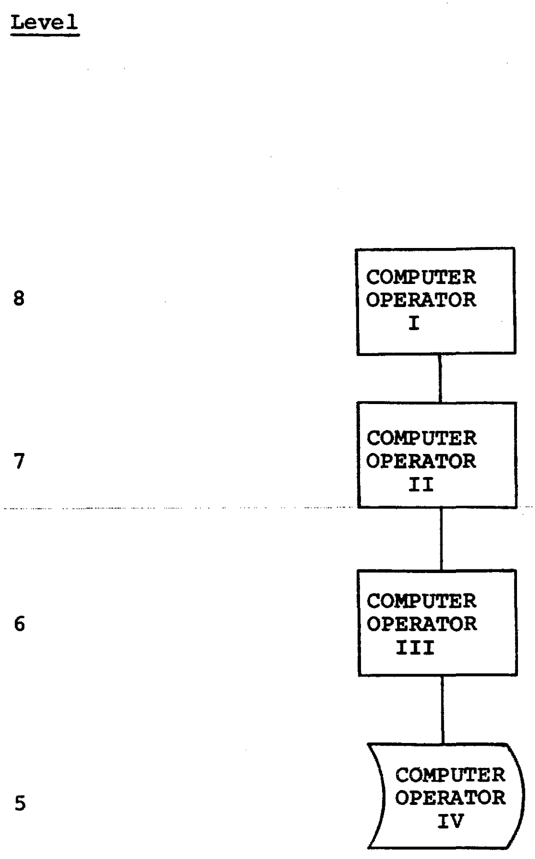
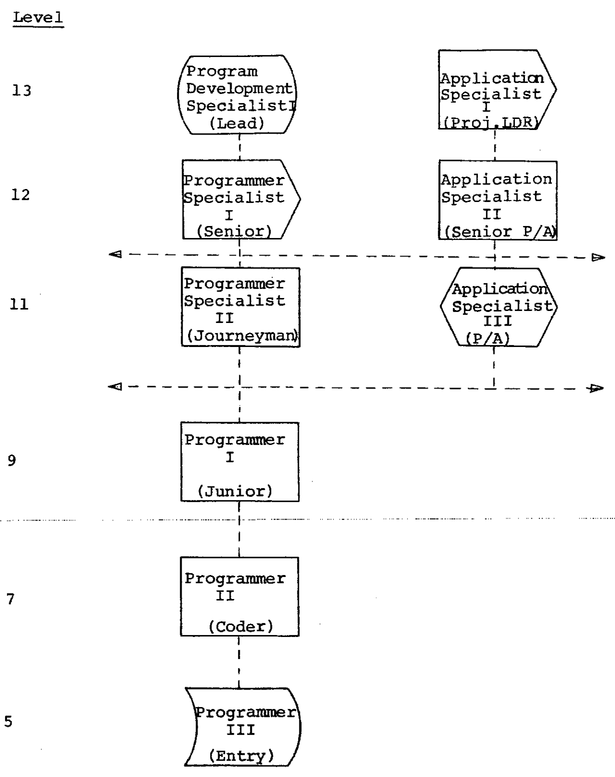


Exhibit 4

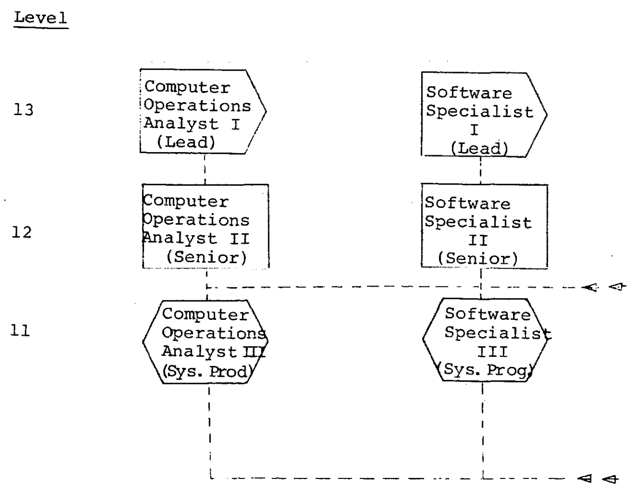
PROGRAMMING FAMILY



Level
13
12
11
9
7
5
Program Development Specialist I (Lead)
Application Specialist I (Proj.LDR)
Programmer Specialist I (Senior)
Application Specialist II (Senior P/A)
Programmer Specialist II (Journeyman)
Application Specialist III (P/A)
Programmer I (Junior)
Programmer II (Coder)
Programmer III (Entry)

Exhibit 3

TECHNICAL SPECIALISTS



Level
13
12
11
Computer Operations Analyst I (Lead)
Software Specialist I (Lead)
Computer Operations Analyst II (Senior)
Software Specialist II (Senior)
Computer Operations Analyst III (Sys. Prod)
Software Specialist III (Sys. Prog)

Exhibit 5

EXAMPLE OF TRAINING PROGRAM FOR A TYPICAL CAREER MOVEMENT FROM THE COMPUTER AID FAMILY (A)

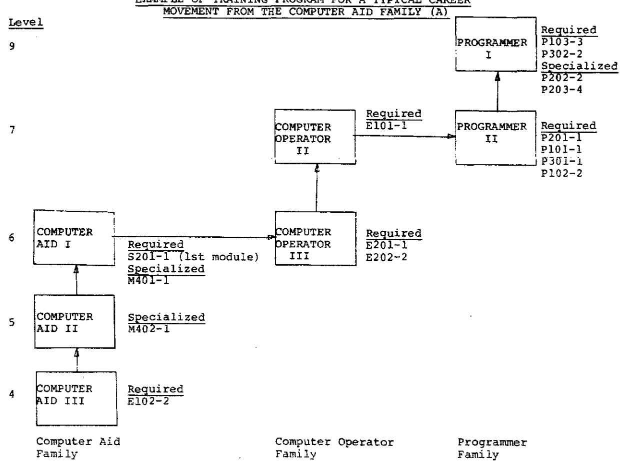


Exhibit 6

SAMPLE CONTENTS OF THE COURSE CATALOG

Category and Major Topic	Course Code and Title	Page
Programming		
P1 Methods	P101-1 Programming Standards	22
	P102-2 Debugging Techniques	23
	P103-3 Program Optimization Techniques	24
P2 Application Languages	P201-1 Computer Programming	25
	P202-2 Assembler Language Coding	26
	P203-4 RPG Coding	27
P3 Software Utilization	P303-1 Basic Job Control	28
	P302-2 Intermediate Job Control	29
	P303-3 Advanced Job Control	30
	P304-2 EXEC 8 Control Language	31
	P305-4 IMS - 8	32
P4 Data Communications Programming	P401-1 Communications for Programmers	33
	P402-2 Communications Access Methods Programming	34
	P403-2 BTAM Workshop	35
	P404-2 QTAM Workshop	36
	P405-2 1100 Communications	37

Exhibit 7

Course	P 201-1. Computer Programming
Scope	Provides a thorough knowledge of hardware concepts, a working knowledge of programming in Assembler and COBOL languages, and a basic knowledge of Job Control language.
Topics	1. History and Concepts of Data Processing 2. Programming Fundamentals 3. Introduction to S/360 and S/370* 4. Assembler Language Coding 5. Cobol Programming 6. Decision Logic Tables 7. Job Control Language* 8. Debugging. 9. Case Studies and Examinations 10. Reference and Standards Manuals* *Separate modules for different equipment manufacturers
Achievement Criteria	Ability to analyze, organize, write, test and debug a program written in assembler and in ANSI COBOL. Written tests provided.
Prerequisites	Entrance test
Course Length	280 hours (7 weeks)

Career development—A new approach to performance appraisal

by WALLACE C. ANDREWS and LESTER J. SONTAG

McDonnell Douglas Automation Company
St. Louis, Missouri

INTRODUCTION

Ours has been called the age of the "knowledge worker." What this means is that people, especially people who work in high technology industries, have become dramatically more important to the success of those industries than ever before. Company executives, who have long talked about their people as "our most valuable asset", are beginning to really believe it. And that is why managers of professional people are becoming increasingly concerned about finding more effective ways to enhance the performance of these highly trained employes.

It would seem to follow that performance appraisal and evaluation is an area loaded with potential payoffs in terms of getting people to do their jobs better. So it would seem. Letting people know where they stand, identifying improvement areas, and giving positive feedback in a straightforward manner are intended to result in upgraded performance. The results, however, frequently fall disappointingly short of the good intentions.

The purpose of this paper is to relate the author's experience in attempting to face up to the attendant problems of performance appraisal in one data processing organization and to share an approach which places the focus on career development.

We would like to say a word about our methodology. Much of the data upon which our development efforts are based has been informally gathered over a period of time. In some instances, it is anecdotal—individuals relating their own personal experiences. In other instances, we are drawing upon planned feedback sessions where the ideas and opinions of managers and supervisors were directly solicited. Throughout the process of research and experimentation we have relied heavily upon some of the well documented findings of behavioral sciences, and upon our own intuitive ideas about what contributes to productive human interactions. What we are offering is neither panacea nor an impressive set of statistics. Instead, we want to share an evolving process that points rather convincingly, from our perspective, to the kinds of issues that need to be considered if we are serious about helping people realize career goals.

DISSATISFACTIONS WITH PERFORMANCE APPRAISAL

The initiative to "do something about" improving our performance appraisal process started with the familiar dilemma. It went something like this: "We think we should be conducting performance reviews, but what is happening often appears to be of questionable value to all concerned." Behavior of managers and supervisors indicated ambivalence. In many instances it took nothing short of edicts and not-so-veiled threats to insure that the task was getting accomplished in some manner. Behavior of workers, on the other hand, was often defensive in the face of "constructive criticism".

Other dissatisfactions were in evidence. The process itself came under fire. The supervisor would fill out a check list type of rating form, ranking a number of attributes on a scale of "poor" to "excellent". He would then sit down with the individual to explain or "justify" the reasons for each rating, and then require a signature at the bottom of the form to indicate the "discussion" had occurred. The expectation of both parties was that the points covered would tend to center on the negatives. From the supervisor's point of view, this was the way to get at better performance. But as the employe saw it, any identified performance problem might represent a road block to a pay raise. To him it sounded accusative, no matter how hard the supervisor tried to put it in constructive terms, and he tended to become defensive or say very little at all. To avoid the discomfort, some supervisors opted to place most ratings on the high end of the scale ("all my people are good performers"), or on the low end ("you have to be a super performer to get an excellent from me"), or right down the middle ("everyone is about average"). The result, of course, was an impersonal approach that elicited little commitment from either party. And ironically, the rating approach of the supervisor quite often revealed more about his management style than it did about the employe he was evaluating.

A further difficulty for the supervisor was that he was attempting to deal with professional people with an instrument that failed to take into account professional issues.

KNOWLEDGE:	To what extent do you demonstrate a knowledge of the available techniques, capacities and limitations of your job?
COMPLETE PICTURE:	To what extent are you aware of the full picture; how does what you are doing fit in with the total projects?
GOOD JUDGMENT:	To what extent do your decisions and actions tend to work out well?
INTERPERSONAL COMPETENCE:	How well do you think you get along with people with whom you work?
INITIATIVE:	How self-directed are you?
PROFESSIONAL CURIOSITY:	To what extent do you approach things inquisitively and analytically?
PROFESSIONAL PRIDE:	To what extent do you like to display what you have done?
GROWTH ORIENTATION:	To what extent are you interested in growing, changing?
TENACITY:	To what extent are you willing to reexamine the subtleties of the problem?

Figure 1

There was no built-in goal setting or career planning. There was no built-in stimulus to discuss the job itself or the individual's ideas or feelings about various aspects of his job. Instead, the supervisor found himself in the position of "playing God", as Douglas McGregor put it, attempting to judge the worth of his fellow man. Some of the criteria for rating related more to personality that to performance. Or, they had more the flavor of the production line than of programming.

FIRST STEPS

It appeared that one of the first steps was to tackle the rating form itself. Requests were sent out to companies all over the country for samples of appraisal tools currently in use. We asked the question, "What don't you like about the form you are using?" Objections were listed so that problems experienced by others might be avoided. One of the frequent problems noted was where salary discussion was linked with performance appraisal and development planning. We had already discovered that the money issue inevitably dominates and intefers with other concerns, so we elected to make it a separate discussion.

Since we also wanted to be sure that any performance criteria used would relate as concretely as possible to the data processing professional, obviously we felt we could not rely totally upon the experience of others. So we addressed some questions to our own organization. What do top-notch, productive programmers have that the lesser performer does not have? What are the significant characteristics that would help you know if a programmer was a productive professional? These questions were raised with a cross-section of various experience levels and managerial levels through a series of individual interviews. Included in the interviews were programmer trainess, veteran programmers, supervisors and managers of programmers, and in-house customers—persons who regularly utilized the services of programmers. In each case, the responses were recorded in unedited form. Approximately 150 persons were interviewed and all responses were compiled to determine commonalities. From this process a list of nine criteria emerged (see Figure 1).

This now reflected the thinking of persons who are presumably best equipped to know the job and who are directly involved with programmer performance.

We now had criteria. What more was needed? From our own experience, as well as from evidence reported by others, we knew that having valid criteria does not automatically eliminate the negative results that frequently grow out of appraisal discussions. Because of this, it was clear that serious consideration had to be given to the manner in which the appraisal process was undertaken.

We started out with several basic assumptions about people. We accepted the theory that people are essentially growth oriented and will strive to meet this need in ways that make sense to them. They are interested in work and want challenge and responsibility. We also ascribed to the concept that people will tend to develop commitment to problem solutions and growth goals which they have an active role in defining.

MODELS OF INTERACTION

It occurred to us when we looked at the traditional appraisal interview in terms of a model of interaction that the model itself violated our own basic assumptions about people. In some ways it suggested a Doctor-Patient Model. The supervisor (doctor) was in the position of diagnosing the employe (patient) and then prescribing a remedy or remedies to correct deficiencies. As mentioned earlier, appraisal discussions often centered around the things that are wrong (the patient's ills). And of course, the burden of responsibility for having good data for diagnosis and the right answer for remedies rested with the supervisor. In this role, he could understandably assume that the employe often disliked what he heard and resisted doing what was "good for him".

The appraisal interview also tended to take on the characteristics of a Judge-Accused Model. It usually involved the communication of subjective judgments by a person who had a great deal of influence over the fate of the employe. It is not difficult to see why he would rise to defend himself. If the criticism felt like judgment, and if that criticism was suspected of being biased and incomplete, then the individual's receptivity to any plan of corrective action was correspondingly low. Small wonder that concern about salary tended to dominate the appraisal discussion.

Clearly, a more facilitative model was required. First of all, it should take into account the interest and desire of the individual for growth and development. His ideas, opinions and knowledge of the job situation must be considered as having validity and value. A second need was for generating more commitment. Since commitment comes through full participation, all aspects of the appraisal process should be interactive, culminating in a mutually agreed upon plan for development. And finally, while some attention must be given to history, what has occurred on the job, the main thrust should be future oriented, building the future on past identified strengths.

A Systems Model, in which the participants are viewed as collaborators, most closely approximates the final design. In this model, appraisal and development planning are not viewed as something one person does *to* or *for* someone else. Rather, it emphasizes a dynamic process of *joint effort* and *shared responsibility*. The focus is on movement, from where we are to where we want to be. And the movement is not based on one-sided perspective, coersion, or reaction to problems. On the contrary, it is goal-directed and proactive. A premium is placed upon clear definition of the job situation, data gathering, and feedback, action planning, implementation and follow-up.

The Systems Model concept provided the essential structure which we attempted to incorporate in what we now call our Career Development Review System (CDRS).

FIELD TEST

To test our idea, an experimental form was introduced in a programming department and used for a period of about one year. The initial response was mixed. The process represented a radical departure from what people were accustomed to and some did not fully trust it. There was resistance to the amount of effort required, particularly on the part of the supervisor. But as the system recycled (in this case, in six months), favorable responses increased, and the quality of data generated in the review sessions improved. As people began to accept the stated purpose for the discussions and to recognize the values associated with the entire process, supervisors became more enthusiastic. At the conclusion of the test period, feedback indicated we were on the right track. After making revisions based on the trial experience, a presentation of the System was made to a top management team. It was then introduced to all managers and supervisors through a four-hour orientation and training session. In order to increase their involvement with the System, we asked for any ideas and suggestions they might have to improve the review form. Further refinements were made and we were ready for implementation.

APPROACH AND CONTENT OF CDRS

The collaborative character of the CDRS begins with the manner in which the forms are prepared. In traditional performance reviews, the employe enters the interview empty-handed. With the CDRS both the employe and the supervisor prepare identical forms, each bringing his own written comments to the sit-down session. The employe is expected to provide major input to insure information flow from both directions. The stated purpose of the discussion is to gain mutual understanding, to explore and resolve areas of difference, and to generate data for effective decision-making. Candor is encouraged in the hope that accusative and defensive behavior can be reduced or avoided.

The first issue addressed in the CDR discussion is the job itself. Question 1 asks, "What are your most important responsibilities as you see them? Explain what your job involves as fully as you understand it." The question is intentionally personalized. What are *your* job tasks and how do *you* prioritize them? Formulating an answer to this question might be the first time an individual actually tries to define what he does. Through this process he may discover areas of uncertainty or vagueness. It may stimulate a deeper discussion with his supervisor to clarify, reshape or even redesign the job. Part of this discussion might also deal with the vital question, what is your contribution. In other words, does the individual perceive the job as worth doing. And what does he feel he brings to the job that is uniquely his.

Discussion of the job is a rewarding and a demanding exercise for the employe and the supervisor. Both parties lay out their expectations. Points of disagreement become important areas for negotiation. In some instances, the employe's way of approaching the job may result in his inventing his own job description and the basic performance parameters relating to that job. If the supervisor is flexible enough he may recognize that while his own views differ to some degree, he is willing to modify them for the sake of encouraging creativity. On the other hand, he may discover that the employe simply does not have sufficient or accurate data. His perspective is too narrow. The supervisor then has the ideal opportunity to discuss the "big picture". Or, to put it in different terms, he is able to perform the vital management function of removing obstacles which hinder performance. He removes the problem of limited information and enables the employe to take a fresh look. In the role of collaborators, both are working to solve a common problem. And in so doing, they are able to bring their views into harmony with the integrity of both still intact.

The CDRS "blank page" approach to job definition does not resolve the issue of performance standards in the sense of setting a base line for everyone with a similar job title. It provides no ready answer for the data processing professional who wants descriptive guidelines on job tasks. What it does do, however, is offer a process whereby job tasks are defined for each individual situation. The advantage is flexibility. Changing requirements through time, as well as differing organizational unit and individual needs, can readily be accommodated.

The second question raised in the CDRS form opens the door for further obstacle removal. The employe is asked, "What portions of the above mentioned job responsibilities do you not like?" At first blush, this sounds like opening the door to a gripe session. Every position has distasteful aspects and no one is totally pleased with everything he has to do to get his job done. So why bother? Why stir up the inevitable moans and grumbles?

Let's go back to basic assumptions. We started out by affirming that most people are interested in work and want challenge and responsibility. If there are dissatisfactions about the job, we would then assume they are likely to be things that make it impossible or at least very difficult to do work at a meaningful level of performance and accomplishment. In other words, what sounds like moans and grumbles may be more than that. Some common "dislikes" that might emerge are such things as unrealistic demands on schedule, conflicting orders (too many bosses), red tape, feelings of inadequacy due to lack of training in a specific area, uncooperative fellow workers, heavy demands on elements of the job that seem to make little or no real contribution, inadequate facilities, etc. Are these problems worthwhile to discuss or not? Do they have impact on performance and professional development or are they trivia? We believe they are part of the data gathering/fact finding step essential to a true systems approach.

To be sure the supervisor does not always have it within his power to remove every obstacle impinging on employe performance. (And it might be added, neither does the employe expect him to be super*person.*) On the other hand, the supervisor may be operating with a "skewed deck" if he is not fully appreciative of how the employe perceives his own situation, whether that perception is accurate or not. It is conceivable that some of the obstacles may go away simply by exposing them. For example, what the employe sees as unrealistic demands on schedule may be the result of confused priorities which his supervisor can help straighten out. In some cases, the obstacle might be converted into a challenge by raising the question, "How can *we* overcome this?" Receiving the support and interest of his supervisor may be all it takes to help the employe tackle the problem, with the potential payoff of higher performance effort. In those cases where an obstacle cannot be removed or reduced, the employe's needs are satisfied if he knows that his supervisor will go to bat for him on other problems that can be handled.

The next discussion question which the CDR form confronts is the concern for growth and development within the context of the current job. "What additional responsibilities would you like to include with those you now have?" At issue here is maximizing the individual's contribution and utilizing as fully as possible the talents, skills and abilities which he brings to the job. For some it may be the opportunity to "manage" more of their work. For others, it may be more interface with outside clients. In any event, it begins to focus on the career interests the individual would like to pursue. It also begins to suggest the kinds of developmental activities and plans appropriate for joint consideration.

There is an interesting sidelight to the question on additional responsibilities. It has to do with a useful distinction between two types of motivation. An employe may be considered motivated if he wants to do a good job, at least good enough to stay out of trouble or to be recognized as a "steady performer". He manages to remain reasonably safe from criticism and, therefore, reasonably safe so far as job security is concerned. A second type of motivation is reflected in the individual who tries to accomplish beyond safe levels. He is a risk-taker, who extends himself to produce at a high level over a period of time.

The CDRS attempts to cultivate a climate that reduces the concern for safety. If there is clear understanding of the job and the nature of the individual's contribution by management and the employe, serious efforts to remove obstacles to performance, along with exploration of ways to enhance his worth to himself and the organization, then there is likely to be more expressed interest to tackle new tasks and new challenges. Employes will learn to take risks because they will see this as the way to goal attainment and career satisfaction.

Questions four and five of the CDRS form turn attention to accomplishments and feelings about past performance. "What job related things have you done well?" and "What job related things have you done least well?" By this point in the development review discussion, a good foundation has been established for having an open exchange on identifiable strengths and weaknesses in the way job responsibilities have been carried out. It is important to note, however, that this "backward look" is not designed to produce a "grade" or to assign a rating. Rather, it is to be viewed as part of data-gathering which leads to action planning. (When we present the CDRS to our management people in training sessions, we emphasize this by referring to the entire CDRS form as a "worksheet" to assist with career planning and goal setting.)

Example of the kinds of questions that have grown out of these areas include the following: Which job related activities have given a special sense of accomplishment? To what extent was the good performance due to skills and abilities, and to what extent was it due to determination and hard work? In the case of the job related things done least well, what were the expectations that were not met? Are these expectations realistic, or should efforts be concentrated in areas more appropriate to the individual's interests and abilities? What involvements are considered important that are not listed among the job responsibilities? If so, should they be?

Reference has already been made to the list of criteria which was developed to aid in the assessment of the way the data processing professional operates on a day-to-day basis. The CDRS form asks for commentary on each item. What, for example, is the behavioral evidence that demonstrates his interpersonal competence? Or, in what ways do his decisions indicate the presence or lack of good judgment?

It might be argued that the list of criteria are subjective factors for which there are no clearly definitive standards. The authors agree. For this reason, no numerical or adjectival check list appears in the form. At the same time, we believe the criteria point to important areas of behavior which are observable and which do impact work effectiveness and professionalism. We are asking the individual to look at what he has accomplished and to think about what has helped with that accomplishment. For example, technical skills may be in place, but a programmer's ability to contribute is dependent in a significant way upon his use of interpersonal skills, initiative, the exercise of good judgment, etc. Conversely, we are asking the individual to assess those things that have blocked or hindered his professional effectiveness and to consider how and where he wants to grow if he is to move forward in the attainment of his career goals. What are his development needs? What can he do over a period of time that will satisfy those needs?

Action planning is the subject of the final section of the CDRS form. The question introducing this section is deliberately phrased, "What do you plan to do to build on your strengths, enhance your skills or increase your effectiveness?" Emphasis, obviously, is on making the most of what the individual has going for him. Here we concur with management consultant Peter Drucker, who advocates utilizing a person's performance strengths and neutralizing his weaknesses. What has he done well? What, therefore, is he likely to do well? And, what does he have to learn to be able to get full benefit from his strength? Again, we encourage *pro*action instead of *re*action.

The "what" question is asking for clearly defined goals and objectives. This means they should be specific, measurable, realistic items. All the preceding discussion concerning job responsibilities, obstacles to good performance, the individual's contribution, his interests, strengths, weaknesses, etc., has been a process of data gathering and feedback. With this shared information the employe and his supervisor should be in a position to zero in on development objectives. They may include such things as skill training, educational needs, projects designed for growth and exposure, special assignments, transfer to a new area, or anything else that would address specific growth needs. To be avoided are the "to get better in every way" kind of statements which sound good but cannot possibly be carried out. The form aids in guiding the planning discussion toward specificity by asking *what* the individual is going *to do* and *when.* A date is then established for a progress review at the end of a period that is appropriate for the particular situation.

BY-PRODUCTS

The CDRS has been in use in our organization for less than a year, so it is still early to make a conclusive analysis. We have, however, experienced some by-products of the system which we feel validate the process and strongly indicate we are moving in the right direction. For example, there is currently a request to expand the use of some version of the CDR system to hourly employes. While the content may be modified, the elements of self-appraisal, collaborative problem-solving and action planning are things that have elicited favorable response and will be carried over. The CDRS has stimulated regular career development planning discussions between individual employes and their supervisors that were previously occurring sporadically, or in some cases, not at all. We believe that while management has some responsibility for stimulating career development, it is a shared responsibility. The individual must take an active role. The CDRS encourages the idea of joint ownership. Copies of action plans sent to our Human Resource Development Staff have an added payoff; they serve as a potentially important source of data for identifying development needs common to groups of people and, therefore, suggest training and development programs that might be offered in-house. It is our suspicion, though undocumented, that the CDRS is helping our managers do a better job of working with their people. But we also readily acknowledge that it has highlighted the need for additional training in the skills of coaching and counseling. This need was anticipated to the extent that a four-hour training session was a requirement for all managers and supervisors prior to implementation. But this provided primarily an orientation with minimal skill training. A full-blown coaching and counseling program is now in the planning stages.

SUMMARY

Changing conditions in business and society are demanding that we develop creative ways to utilize and challenge our people. We believe that redirecting the focus of traditional performance appraisal is a step in the right direction. The process evolving through our Career Development Review System recognizes several key concerns that must be addressed:

1. Individual responsibility for career development must be encouraged in ways that make sense to him/her.
2. More attention needs to be given to building on strengths and uniquenesses as opposed to traditional preoccupation with overcoming weaknesses.
3. Reaching goals and attaining career satisfaction is maximized when there is a climate which encourages risk-taking behavior.

We'd not like to pretend that the CDRS is the full answer to any of these. It is not. We know that further refinements are needed. Some of these will come about as we gain more experience. But there is also the need for continuing exchange of ideas and experiences with others in the field.

BIBLIOGRAPHY

Addison-Wesley Series on Organization Development, Addison-Wesley Publishing Company, 1969.

Drucker, Peter F., *The Effective Executive,* Harper and Row, 1967.

Brynildsen, R. Douglas and Marian A. Kremel, "The Career Development Workshop," Paper Presented for *NTL Conference on New Technology in Organization Development,* 1972.

McGregor, Douglas, "An Uneasy Look at Performance Appraisal," *Harvard Business Review,* May-June 1957.

Meyer, H. H., E. Kay, and J. R. P. French, Jr., "Split Roles in Performance Appraisal," *Harvard Business Review,* January-February 1965.

Thompson, Paul H. and Gene W. Dalton, "Performance Appraisal: Managers Beware," *Harvard Business Review,* January-February 1970.

A systems approach to career development—Report of two surveys

by PAUL D. OYER
U.S. Bureau of the Census
Suitland, Maryland
and
DOROTHY L. RAY
General Research Corporation
McLean, Virginia

INTRODUCTION

The so-called profession of computer data processing has barely reached adolescence. Like the electrical engineering profession of 25 years ago we have no standards for describing our job tasks (what we do), nor for defining job skills needed (how we do it), nor for defining educational needs (what we need to know), nor for estimating time and cost for a complete job (how long it takes and level of skills/knowledge needed).

As a consequence we are seriously hampered in achievement of cost effective use of our computers and even of our people's own time and energy![1] The Practicing Computer professional does not know where to obtain guidelines on what to do and to learn to assure continuing growth and usefulness to his chosen profession and to his employer.[2,3] Employers do not know how to define realistic expectations of their computer professionals,[4] nor what a professional should know or achieve for advancement to the next level in his career, nor even what a reasonable career structure should be for computer professionals.[5]

Many published works have identified all these problems.[6] Some have even made a little progress toward some potential solutions.[7,8] In the educational arena, the ACM has published model curricula for college education in computer science and information analysis/systems design in CURRICULUM '68 and CURRICULUM '72.[9,10] These have proven useful for certain entry level jobs in the computer field, and even provide a sound foundation for productivity and advancement. But advancement to what?[11] What is the career path?[12]

What can we do for the practicing computer professional?[13] Can't we provide him with a career structure and continuing career education guidelines?[14]

To this end, in January 1973, a Special Interest Committee on Career Development was formed by the Washington, D. C. Chapter of ACM with an assist from ACPA.*

* The group was merged in October 1973 with the Special Interest Group on Career Development of the Federal ADP Users Group (FADPUG).

The objectives of this committee are to:

1. Collect and disseminate information on Career Development for Computer Personnel (including surveys, bibliographies, etc.).
2. Identify and propose a model Career Structure and Career Guidelines for Computer Personnel.
3. Identify and Propose Model Job Descriptions for the various levels of the Career Structure.
4. Try to Identify and Propose a Model for Educational, Skill and Achievement Criteria Needed for Effective Performance at each Career Level.
5. Identify courses and curricular models needed for continuing career development and determine where they may be studied—college, in-house or elsewhere.
6. Determine status and usefulness of certification, testing and licensing of computer professionals.
7. Submit models as proposed above, to panels of managers and professionals, computer societies, government organizations, employers, college and standards groups for evaluation and possible validation and inclusion in industry-wide standards.

Some progress has been made toward objectives 1, 2, 5, and 6. In order to achieve any progress at all on objective 4 and to form a more solid base of data for enhanced progress on objectives 1, 2, 5 and 6—it was decided that surveys were needed .While some interesting survey results have been reported, it was deemed essential to identify those most relevant to our stated objectives.[15,16] It soon became evident that we must design and conduct some surveys of our own in order to get data which we could correlate directly to the objectives of our systems approach to career development.[17]

The first two surveys of an ongoing questionnaire survey program have been designed, tested and have yielded valuable data. They have been analyzed sufficiently to report on preliminary results below. We have been most pleased that both professionals and employers have been so cooperative in all our data gathering efforts. Many people seem to be interested in participating in an effort to establish

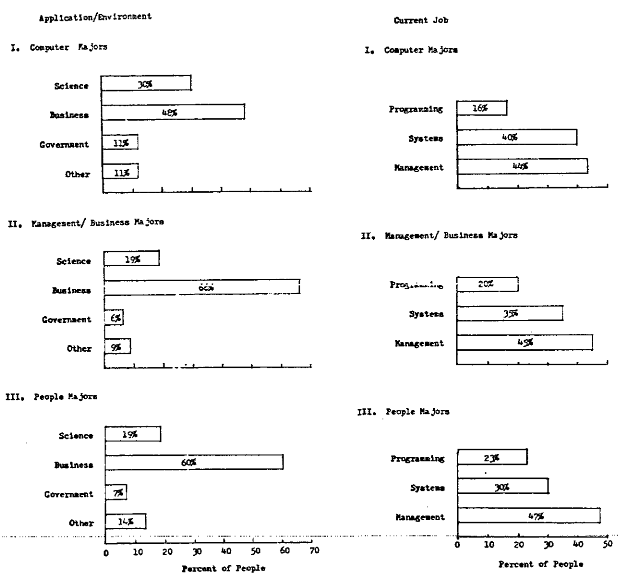


Exhibit 1

our field as a true profession and to get improved performance from our computers and our people.

SUMMARY OF SURVEYS

Two extensive surveys have been conducted:

(1) A survey (called Survey 1) of the formal and continuing educational qualifications of 509 persons attending a number of professional meetings of computer personnel. The meetings were sponsored by ACM, ACPA, AFIPS and a government organization. The survey identified the major disciplines studied in college, and the number of formal credits and short courses taken in six (broad) categories of knowledge. It sought to identify the types of sources of computer skills, and the percentages of job-related skills derived from each source-type. Several interesting results have been obtained.

(2) A survey (called Survey 2) of identification of educational needs of computer personnel as viewed by their employers. A total of 103 organizations have responded so far on behalf of 10,060 computer personnel in five major job classes. The organizations represent small to medium non-government installations and small to large government installations. This survey requested the employer's view of education needed for the job in areas of knowledge treated in 14 graduate courses and an opinion of the quality of their EDP employees.

Both surveys were conducted under sponsorship of the SIC/CD of Washington, D. C. Chapter ACM with an assist from ACPA. Survey 2 was distributed to government organizations under the auspices of the Federal ADP Users Group (FADPUG).

THE SYSTEMS APPROACH

Results from these and other surveys and data from other sources will be constantly fed into the Career Structure effort of Nan Ayer's group[18] into the Certification and Testing Research of David Skeen's group[19] and to the ICCP, into Bob Henry's MIS Research Center,[20] into Bob Meyer's education source file, and to any other groups who want to unite with us to work toward our professional and career development objectives.

Additional surveys may be needed to get more detailed data in areas already covered and to gather data in new areas. We are prepared to do so, as needed. We know we will need to acquire task and job analysis data, but understand AFIPS has a publication coming out on this, based on Ray Berger's efforts.

We welcome help from, exchange of data and cooperation with, any person or group who want to work toward establishment of professional standards.

SURVEY 1

Educational profiles

A survey called "EDP EDUCATION SURVEY" was designed to gather a profile on formal and continuing educational backgrounds on a sizable sample of computer professionals. What better way to select professionals than choosing attendees at professional meetings!

So 509 EDP people filled out one-page questionnaires at five meetings—sponsored by AFIPS, ACM, ACPA, and a large government agency—in New York and Washington, D. C.

Most of the results have been tabulated and some of them are ready to report (although statistical tests of significance are yet to be computed). Several interesting factors were covered, and can lend some insight into the knowledge level of practicing computer professionals today.

Of the 509 professional subjects, 84 percent have bachelors degrees, 47 percent have done postgraduate work, 36 percent have masters degrees, and 6 percent have doctorates.

As to work environment, 47 percent are in business data processing, 29 percent in scientific, 7 percent in education and 17 percent other. Thirty-two percent are programmers, 31 percent are in systems, 26 percent are managers, and 11 percent other.

For a sample of 128 attendees at the first National Computer Conference in New York in June 1973, 40 percent of the attendees responding were from New York, 22 percent from the greater Washington, D. C. area and 38 percent elsewhere. Thirty-three percent were managers, 23 percent programmers, 16 percent educators, and 25 percent systems people.

Of the Educators, 19 percent had doctorates, 57 percent had masters and 73 percent came from out of town. Thirty-seven percent of the managers had masters degrees as contrasted to 24 percent of the programmers. Fifty-three percent of the managers were in a business DP environment and 21 percent were in government. Over 90 percent of all government attendees were managers and of the business DP attendees, 45 percent were managers. Of the Scientific DP attendees, 50 percent were programmers.

Of the entire survey population of 509, only 8 percent hold the CDP (Certificate in Data Processing) and two percent wondered what it is! Forty percent would be interested in getting a CDP. As to college majors, 23 percent were mathematics, 14 percent computer science (or related), 12 percent business or economics, 10 percent physics or engineering, 7 percent management majors, 6 percent social sciences, 3 percent each in psychology, education, philosophy, and statistics plus 16 percent other majors.

Skills needed in technical and peopleware areas

This survey requested credit hours and short courses taken in 6 major knowledge and skill areas:

$$\left.\begin{array}{l}\text{1. Computers}\\ \text{2. Systems}\\ \text{3. Models}\end{array}\right\}\text{ Technical}$$

$$\left.\begin{array}{l}\text{4. Organizations}\\ \text{5. People}\\ \text{6. Social \& Economic Impact}\end{array}\right\}\text{ Peopleware}$$

The first three areas are traditionally recognized as essential knowledge areas for a computer professional. These are areas we all struggle in to try to keep pace with the state-of-the-art. But what do we know in the last three essential areas? Apparently "very little" for most of us. The results of this survey by SIC/CD of 45 diversified computer people at a large government bureau disclosed that the average number of college credits in the three technical areas (accumulative) was 20 credit hours (semester) as contrasted to only six hours in the "Peopleware" areas. Most of those credits were received by the most recent college graduates.

An ACM/NBS technical symposium

A special analysis was performed on 245 survey forms completed by attendees at the 12th Annual One-Day Technical Symposium of the Washington, D. C. Chapter ACM held at the National Bureau of Standards on June 21, 1973.

Significant results noted: (See Exhibit 1)

"Computer" majors currently work in scientific (including Government) environments 41 percent—more than all other majors—25 percent.

Management and business majors tend to work in Business Data Processing—66 percent.

More computer majors—40 percent—are currently engaged in systems jobs than other majors. Computer majors have more advanced degrees—79 percent masters and 17 percent doctorates.

More "people" majors (including Psychology, Sociology, Political Science, Anthropology, Personnel, Training, and Philosophy) are currently engaged in programming—23 percent—than are others.

Other results worth noting:

1. A high percentage 47 percent of non-mathematicians are interested in getting a CDP. Only 14 percent of math majors are interested.
2. Most people holding a CDP already have a degree.
3. Four times as many CDP holders are in systems as in programming.
4. Almost all those who studied management, are in management. Most EDP managers did *not* study formal management courses in college.
5. Of attendees with Doctoral degrees, 40 percent are in management, 28 percent in systems, 20 percent in education. 60 percent of the doctors are in scientific applications, only 26 percent in business.
6. Of attendees with no degrees, 68 percent are in business applications, only 29 percent are in scientific.
7. 101 have math majors, 37 Computer Science, 30 Physics, 22 Business, 20 EE, 16 English, 16 Chem., 17 Management, 17 MIS, 17 Econ., 13 Social Studies, 12 Educ., 9 Philosophy, 7 Psychs., 7 Statistics, 4 Anthropology, etc.

Other Highlights:

Computer majors (at least those who attended this computer symposium and filled out the Survey Form) have a significantly high percentage of advanced degrees.

16 percent have Doctorates
83 percent have Masters Degrees

For Mgt./Bus. Majors:
57 percent have Masters but only 2 percent have completed Doctor's degrees (although 22 percent have have studied beyond the master's level).

For "People" Majors:
51 percent have Masters degrees and 5 percent have doctorates (but only 12½ percent have pursued formal study beyond the master's level).

Of the 245 respondents, 51 (or 21%) inquired about getting a copy of the results of the attendees' Education Profiles.

Those who were curious enough to inquire about attendee EDP'ers Education backgrounds, have better formal educations than the average symposium attendee!

	Masters Degree	Doctorates
Inquirers	35 percent	11 percent
Non-inquirers	28 percent	7 percent

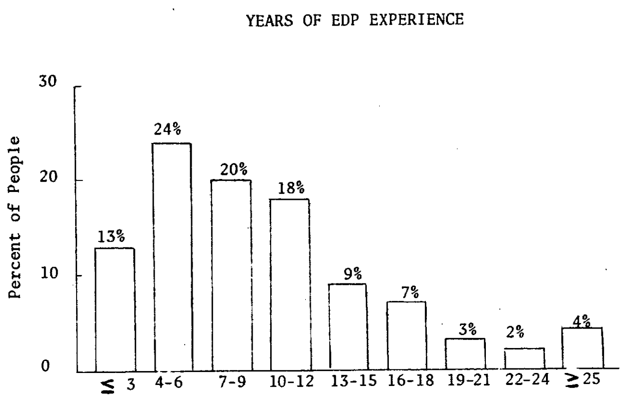


Exhibit 2

This may mean they are also more curious about pursuit of knowledge and persist until degree and other goals are achieved.

Of those with management majors in MIS/EDP, 100 percent were inquirers. Of management majors in non-EDP subjects, only 4 percent were inquirers!

Percentage of "Inquirers" for various majors:

Major	Percent Inquired
MIS Mgt.	100 percent
Engineering Science	75 percent
Statistics	42 percent
Biology	42 percent
Accounting	40 percent
Physics	35 percent
Business	32 percent
Management	4 percent
Economics	0 percent
MBA Degrees	0 percent

A higher percentage of systems people (25 percent) are curious about EDP'ers educational profiles than are programmers (13 percent) or even practicing managers (19 percent).

Most attendees (57 percent) have less than 10 years of EDP experience. Nine percent have over 18 years experience and 13 percent have worked in EDP less than four years. (See Exhibit 2)

A detailed analysis was made of profiles of 70 attendees at an ACPA professional seminar. Most attendees (60 percent) thought they acquired their EDP skills on the job or through experience rather than from formal education or training. (See Exhibit 3) Apparently the educators and trainers still have a big job ahead of them!

Some special comments received from the total population of 509 respondents are worthy of special mention. They would like to see:

More university courses in technical and peopleware areas.

More in-house courses of an education (not merely training) nature.

More short courses in these critical areas by universities and others.

More career planning and guidance by employers.

More guidelines by professional associations for self-guidance.

New Continuing Education Plans for the Computer Professional.

Establishment of Professional Tests and other Achievement Criteria for promotion to various levels.

SURVEY 2

The questionnaire for EDP managers

A survey called "CAREER DEVELOPMENT RESEARCH SURVEY (For EDP Organizations)" is being conducted. Preliminary findings are reported here.

Addressees of the questionnaire were told—"Why should you help this project? Its purposes are: (1) to identify needs for specific skills and knowledge at various levels of the EDP field, (2) to propose a workable multi-level EDP Career Structure (Career paths, ladders, etc.), (3) to translate skills and knowledge needed into specific requirements in courses, experience, and achievement criteria for each EDP career level."

Key EDP Managers were asked to respond on behalf of their organizations and to provide their personal view of educational needs of computer personnel on the job for five different job classes.

Respondents from 56 government installations reported for 6934 EDP employees and from 47 non-government installations for 3129 EDP employees. The respondents individually average 12.7 years of EDP experience with a range of two to 36 years.

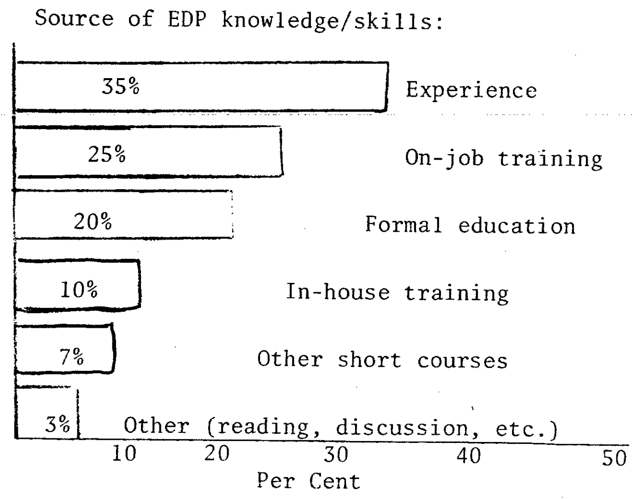


Exhibit 3

A non-government sample

A randomly selected sample of 20 non-government installations was analyzed in depth. These respondents themselves were all managers and by background 45 percent were Business majors, 25 percent mathematics majors, 15 percent EDP majors, and 15 percent other. Their processing environments were 78 percent Business, 16 percent Scientific, and 6 percent other. Their organizations were 35 percent businesses, 5 percent Scientific. 30 percent service bureaus, 20 percent non-profit, 5 percent computer manufacturer and 5 percent software consulting.

Of the 1539 EDP people they responded for, 11 percent were Supervisors, 11 percent Systems personnel, 16 percent Programmers, 48 percent Operators, 12 percent Support and 2 percent other. The respondents representing 63 percent of the Supervisors state that Supervisors are "professionals," 61 percent for Systems professionals, 58 percent for programmers as professionals, and 10 percent for Operators as professionals.

Detailed findings

Some of the questions from the survey are repeated here with the percent of responses shown for affirmative answers to that specific item.

In question four below, 68 percent of the respondees say that they are short of well-qualified EDP supervisors and 63 percent for system personnel. Similarly many EDP personnel are currently under-qualified for their jobs. There seems to be no oversupply of systems analysts and no over-qualified supervisors or system people. On the other hand, over one-quarter of the installations have too many programmers or some over-qualified programmers with no career step to move up to.

4. In your observation or opinion—(Check any item which applies)

	Supervisory	Systems	Programming	Operations
	Percent			
a. Which positions have the shortest supply of well-qualified people?	68	63	21	10
b. Which ones are currently filled by some under-qualified persons?	42	32	21	32
c. Which positions have an over-supply of well-qualified personnel?	16	0	26	10
d. Which ones have some spots filled currently by over-qualified persons?	0	0	26	21

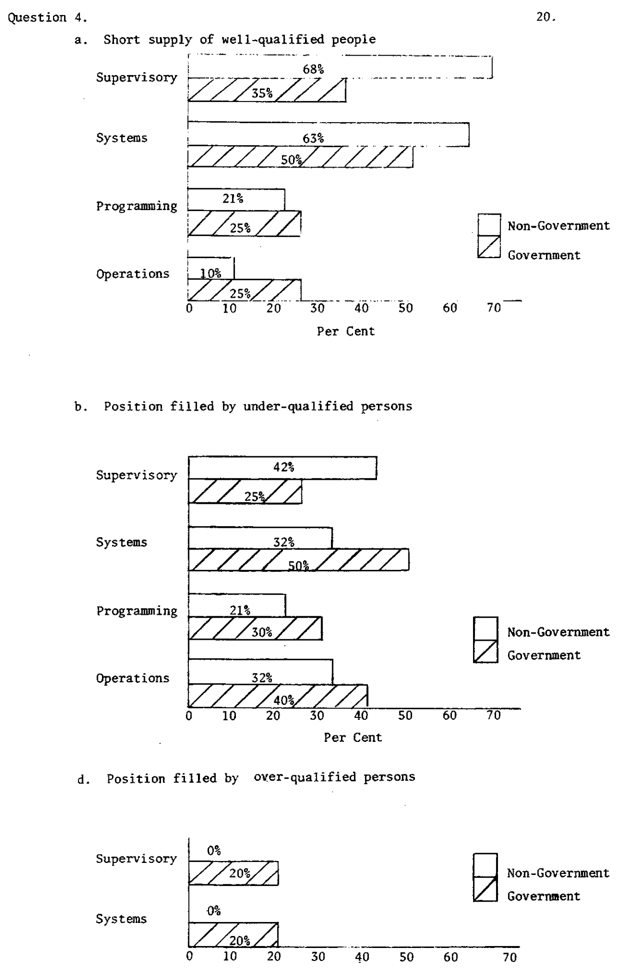


Exhibit 4

In question five below, preliminary analysis shows that items over 40 percent and below 10 percent are significant. Thus EDP supervisors are short on formal management education, human relations, and ability to communicate. Programmers need to learn more about human relations and numerous operators do not have enough technical knowledge to do an adequate job.

5. Where EDP personnel are not adequately qualified for their present or their next position, which areas do you believe are their greatest shortcomings?

	Supervisory	Systems	Programming	Operations
	Percent			
Formal Education	45	20	10	40
Technical Knowledge	15	15	30	75
State-of-the-art knowhow	10	35	10	20
Knowledge of applications	15	30	25	40
Human Relations	65	25	45	25
Ability to Follow Through	35	30	25	30
Ability to Communicate	50	45	35	25
Ability to Lead	35	10	20	25
Creativity and Imagination	30	15	25	25
Social Responsibility	15	5	15	15

In question six, percentages below 10 percent and above 50 percent are significant. Thus programmers and operators should sharpen their technical knowledge and improve their job skills. The "myselfs," on the other hand, already know "all about" cooperation.

6. What do you feel should be some prime objectives of additional training for EDP Personnel? Check as many as apply for the various positions.

In question seven below, respondents listed their recommendations as to which key senior personnel should have the knowledge from 14 specific graduate courses, with a rating of Essential, Desirable and Not Needed. Using a weight of 10 for E, 5 for D, and zero for N, a rating on the scale of 0 to 10 was calculated for each item. Scores show that 11 of 14 courses are regarded as critical knowledge for senior systems professionals (scores of 6 or over).

7. In 1972 the ACM Curriculum Committee on Computer Education for Management (under a grant from the National Science Foundation), recommended a graduate curriculum of 14 courses they identified as needed by EDP Systems and Supervisory personnel. Please indicate which of these courses you regard as *E*ssential, *D*esirable, or *N*ot Needed by certain key EDP personnel. Fill in as many as apply with E, D, or N.

Courses	Myself	Supervisory	Systems
The EDP Systems Life Cycle	6.8	6.8	6.1
Computer Systems	7	6.9	7.7
File and Communication Systems	5	5	8.6
Software Design	5	4.2	8.2
Modeling and Operations Analysis	6	4.6	5.6
Information Analysis	5.7	5.3	7.1
Systems Design	6.7	6.7	9.7
Systems Development Projects & Case Studies	5.4	6.1	8.0
Information Structures	5.6	5.0	6.9
Functions of an Organization	7.2	7.7	6.8
Information Systems for Planning & Decision Making	7.5	6.5	6
Human and Organizational Behavior	8.1	9.0	6.5
Administration of Information Systems	8.1	8.0	5.3
Social Implications of Information Systems	6.1	5.3	5

OBJECTIVE	Myself	Superv's.	Systems	Prog's.	Oper's.
	Percent				
Improve job skills	32	32	54	63	74
Broaden background knowledge	42	58	48	42	37
Increase technical knowledge	27	26	54	63	69
Improve ability to relate to others	27	64	69	47	26
Keep up with latest technology	63	48	63	26	16
Learn better cooperation	10	42	21	53	31
Refresh old skills/knowledge	32	20	27	20	5
Practice Leadership Skills	53	69	26	5	10

Note that this sample covers smaller and less sophisticated computer installations than Bob Henry's.[20] Yet the critical needs (skills useful) overlap in several important areas.

A GOVERNMENT SAMPLE

A randomly selected sample of 20 government EDP installations was also analyzed and compared with the non-government sample. These respondents included managers, computer specialists and analysts. Forty per cent hold masters degrees, 20 percent majored in engineering, and 13 percent each in math, management, English, law, and education. Their data processing environments are 66 percent business-type government processing, 17 percent scientific and 17 percent other.

Of the 2096 EDP people they responded for, 8 percent are supervisors, 13 percent systems, 31 percent programmers, 29 percent operators, 13 percent support, and 6 percent other. Respondents representing 82 percent of the supervisors regard their EDP supervisors as "professionals," 92 percent for systems professionals, 75 percent for programmers as professionals, and 0 percent for operators as professionals.

MORE COMPARISONS

Some significant differences between the government and non-government samples analyzed and compared for Survey 2 are shown in Exhibit 4.

In Question 4a, note that government has the greatest shortage of well-qualified operators while non-government has the largest shortage of well-qualified supervisors and systems personnel. Question 4b reveals that 50 percent of systems personnel in government are under-qualified. In Question 4d, note that 20 percent of both government supervisors and systems people are over-qualified and apparently don't have sufficient promotional opportunities or other adequate challenges.

CONCLUSION AND FURTHER SURVEYS

Further conclusions could be made by interpretation of the data reported in the two surveys above, but we think the reader may wish to make his own interpretations from, the basic data. Ongoing surveys will be continued by our Career Development group. We welcome assistance from, and exchange of data with, anyone else interested in promoting and measuring the degree of professionalism in our field.

We would like to conclude that all levels of computer professionals, including operators, must be updated and upgraded to improve the effective use of our computers and of our people and of their time and talents.

REFERENCES

1. Donati, Frank R., "Computers and Catastrophies," *Data MGMT*, December, 1971. Gives a suggested methodology for selecting programmers who will become assets to the organization. The plan is called SCOPSE. Small Computer Oriented Programmer Selection and Evaluation and encompasses the criteria of programming characteristics, differentiating between ability and achievement, specific abilities and skills, interest, and performance.
2. Brandon, Dick, "Better Systems Analysts a Must," *COMPWRLD*, September 13, 1972. The author believes that a large number of system analysts are unqualified and cites some of the factors which are contributing to this situation.
3. Armer, Paul, "Obsolescence and Self-Assessment," *CPR PROC 72*. This keynote address to the tenth annual Personnel Research Conference explores the problem of obsolescence among computer people and suggests some things which might be done about it.
4. Lassiter, Herbert B., "Improving the Productivity of Systems Analysts and Programmers," *DATA MGMT*, September 1972. Examines the system development process and the resulting task lists for analysts and programmers. In this context, the factors affecting productivity are listed and some tools for improving productivity are suggested.
5. Bride, Edward J., "DP Position Titles 'Chaotic,' " *COMPWRLD*. September 13, 1972. Highlights of a report presented to the ACPA are given and stress the need for standardization of position titles before professional status will be granted to programmers and analysts.
6. Adelson, Richard, "Mediocrity in EDP?" *DATA MGMT*, August 1973. Offers several suggestions for the elimination of mediocrity and the establishment of an exceptional level of EDP personnel.
7. Myers, Dr. M. Scott. "The Human Factor in Management Systems," *J. SYS MGMT*, 22:10-5, November 1971. Job enrichment is discussed as the key to getting the most out of people. Several suggestions are offered for improving the chances of success of management systems.
8. "A Study of Position Titles In The Computer Systems Field," *NTIS*, July 1972. Reviews various existing and historical systems of titling, and considers the pros and cons of defining a structure of position titles by induction from common usage. A formalization of position titles in the computer field is presented, defining the nature and function of these titles.
9. "Curriculum 68," *Communications of the ACM*, March 1968 (Wm. Atchison, Chairman). This is the heavily copied model curriculum in computer science.
10. Ashenhurst, R. L., et al., "Curriculum Recommendations for Graduate Professional Programs in Information Systems," *Communications of the ACM*, May 1972. This is the "CURRICULUM 72" for Information Analysts and Systems Designers.
11. Gluckson, Fred A. and Michael G. K. Flannery, "Development of Computing Professionals," *CPR PROC 72*. Explores the employer's responsibility in developing the computing professional and the steps taken by the National Bank of Detroit in this direction. Describes the in-house training program in terms of resources, facilities, budget, etc., the development of a training program guide, and the administration of the overall program.
12. Cohen, Herbert A., "Planning and Managing Your Career," *DATA MGMT* September 1972. (Conf. issue). Management competence and career mobility depends upon three major factors: the development of new skills and strategies; a refined ability to read complex organizational behavior and the attainment of results.
13. Kaye, Donald, "MIS Career Paths," *CPR PROC, 72*. This research study attempts to determine what career paths are open in cor-

porate life for a Director of MIS. Three case studies are given and seven conclusions drawn relative to the prospects for advancement of a Director of MIS.

14. Kirkpatrick, Curtis B., "Staffing for Computers In The Federal Government," *DATA MGMT*, November 1970. Structured job requirements established by the government for the selection of DP personnel are of great help in selecting qualified candidates.
15. Dickmann, Robert A., "1971 AFIPS Information Processing Personnel Survey," *CPR PROC. 72*. Charts show the distribution of EDP personnel by age, race, sex, highest degree, source of training, years of experience, number of years with present organization, number of employers worked for, number of persons supervised, and salary range.
16. Gilchrist, Bruce and Richard W. Weber, "Employment of Trained Computer Personnel—A Quantitative Survey," *PROC SJCC*, 1972. Summarizes the employment picture for computer personnel as it relates to computer users. Data for government and non-government users and equipment manufacturers are compared.
17. Andrews, Wally C. and Les J. Santag, "Career Development Review System; A New Approach To Performance Appraisal," *NCC '74 Proceedings*, Vol. 43.
18. Ayer, Nancy L., "Step-By-Step: A Career Structure For Systematic EDP Growth," *NCC '74 Proceedings*, Vol. 43.
19. Skeen, David R., "EDP Certification . . . Is It Necessary?" *NCC '74 Proceedings*, Vol. 43.
20. Henry, Robert M., "Skills Possessed and Skills Useful for MIS Practitioners: A Research Report," *NCC '74 Proceedings*, Vol. 43.

BIBLIOGRAPHY

1. Canning, Richard, "Career Programs in DP," *EDP Analyzer* August 1971.
2. Dickmann, Robert A., *Personnel Implications for Business DP*, Wiley, 1971.
3. Fritz, W. Barkley, "Computer System Standards for a Large Organization," *DATAMATION*, February, 1960.
4. Kaye, Donald, "Career Paths in Systems and DP," *J. SYS. MGMT*, June 1971.
5. Kapur, Gopal K., "Sharpen Your Systems Staff Through In-House Training," COMPUTER DECISIONS, March 1971.
6. Oyer, Paul D., "Comprehensive In-House Training," *Modern Data*, 1971.
7. Oyer, Paul D., "Training Business Systems Analysts in Information Systems Design: Use and Evaluation," *ACM SIGCPR PROC*, 1969.
8. Teichroew, Daniel (Ed.), "Education Related to the Use of Computers in Organizations," *ACM COMMS*, September 1971.
9. Weinberg, Gerald M., *The Psychology of Computer Programming*, VanNostrand Reinhold Co. 1971.

EDP Certification—Is it necessary?

by DAVID R. SKEEN

Office of Naval Research
Arlington, Virginia

INTRODUCTION

There is a movement afoot within the computer industry which will have a great impact on EDP personnel. Much literature has been published which concerns itself with EDP personnel certification. The formation of the Institute for Certification of Computer Professionals (ICCP) in August, 1973, has certainly triggered a large amount of discussion within the EDP community. In reality, everyone seems to be for certification and the prestigious distinction of being called "Professional." So it's everyone to the bandwagon! At a glance, it appears that our EDP certification bandwagon is charging ahead—the certification movement is straining at the harness, the bandwagon is creating dust clouds and recklessly moving forward and the computer passengers are frantically waving for forward motion. In essence, everything is moving forward except for one thing, the wheels—and they are going backwards, thus indicating that the certification base has not been firmly established. Is this the true posture of the EDP profession and its certification movement? Before proceeding with a grand and glorious certification program, it might be well at this time to ask a few important questions. One of which might be "Is EDP certification necessary?"

Before addressing this question, we must first identify the problem of which certification is to be the remedy, i.e., is the problem to be solved related to licensure and the protection of the public or is it related to EDP personnel and their betterment? Also, what is the status of the EDP vocation in regards to the "Professional" movement? Are we ready for certification or for licensure? To sum up the entire situation—why certification?

Before answering any question on EDP certification, it will be helpful to define three terms used by the EDP community: base of knowledge, certification, and professionalism. Base of knowledge will be used in the content of this paper to express the basic skills required to satisfy a certain level of a standard job requirement.

The second term, certification, connotes the approving or verifying a base of knowledge that a person should know and understand for his vocation or job. The Webster dictionary states that to certify is "to attest as being true or as represented or as meeting a standard." The key word in this phrase is "standard"... to meet a standard. The question to ask here is ... What standard has been identified to certify an EDP individual? There are several factors which should be considered in certifying an individual, viz:

Testing
Personnel references
Experience
Communications, both verbal and written
EDP contribution
Education

As stated by Harris and Swearingen in their article in *Data Management* of October, 1973, certification is a continual cycle which is designed to establish and maintain a body of knowledge.[1] The steps necessary to do this are depicted in Figure 1. Certification is built upon a body-of-knowledge foundation.

Finally, professionalism warrants a definition. This term is a difficult one to define. Before a definition can be given it will be useful to identify some attributes of a profession, such as:

(1) a defined body of knowledge of high intellectual content acquired by training in depth,
(2) defined standards of competence, and certification that the professional meets those standards,
(3) a code of ethics,
(4) at least one professional society aimed at advancing the welfare of its members,
(5) the responsibility to society to perform in a competent and ethical manner,
(6) the licensing of the members of the profession by the state to practice the profession,
(7) the right and ability of the members of the profession to eject someone from the field for being incompetent or unethical.

For the purposes of this paper, the following will be used when defining the expression "EDP professional": one who is trained in the skills of EDP, competent in the use of EDP tools, orderly and ethical in his approach, and does his work within an established philosophy of EDP. Professionalism involves defining and maintaining a base of knowledge, certifying EDP personnel, and imposing licensure and implementing a policing agent for unethical behavior. Consequently, licensure is the next step after certification has become established. The steps to professionalism are shown in Figure 2.

CERTIFICATION VERSUS LICENSURE

There are two schools of thought dealing with certification. The key question is . . . Should certification include licensure? The primary concern is the public interest and well-being which may be affected by the EDP field and its endeavors. If public health and life are involved, then not only should the EDP professionals be certified but should also be licensed and policed by an agency tasked to protect the public. The ICCP has expressed its case by its title when the expression "Professionals" is used. The attainment of "Professional," if licensure is involved, would require three major phases: identifying a base of knowledge, certification, and licensure. At such an early stage in the development of an EDP profession, it is felt that certification should take high priority since it comes before licensure. In general, the EDP community is not ready for licensure or scarcely ready for certification since a base of knowledge has not been defined or standardized. However, the EDP community must be prepared to eventually answer this question. If licensure is required, then should an individual become "blessed" by an agency before he can practice? Possibly, certification and licensure will evolve in the direction of the Certified Public Accountant (CPA) vice that of the doctor or lawyer. Certification as used in this paper will entail verifying that an individual has obtained a certain level of skills which have been identified via a base of knowledge. Hence, certification is only a portion or part of a profession as defined above.

WHY CERTIFICATION?

Those against certification usually give at least one of the following reasons:

(1) The EDP vocation as a whole is not mature enough to be certified. Standards and skills cannot be identified because of the ever-changing nature of EDP.
(2) To certify means that structure is imposed and, consequently, creativity is hampered.
(3) The majority of the EDP community and the general public is not concerned about the status of being certified.
(4) EDP skills should be considered as a tradesman vocation especially programming since it is becoming easier to learn and use.

The list could be extended to include many more excuses; however, the author does not feel that any of these arguments are particularly valid and only attempt to avoid the EDP certification problem, i.e., identifying a base of knowledge.

It is felt that certification is most urgently needed for several reasons, i.e.:

(1) The critical need for identification and maintenance of a standard base of knowledge.
(2) EDP systems are becoming very complicated and costly. The failure of the EDP vocation to design, develop and implement their systems successfully is a good indication that professional standards and a philosophy have not been identified and used.
(3) Technology of EDP, such as telecommunications, data base management techniques, and sophisticated hardware, has effected the use of computers in almost any imaginable area of our society and is becoming a major industry in our country.
(4) The general public, the users of computer products and services, and EDP employers must be protected from the technical incompetence which has crept into the EDP ranks.
(5) EDP individuals should be responsible for their work, especially when the general public's privacy and well-being are at stake.

From the above discussion, there are three groups of people which would benefit from certification:

the individual being certified
the employer
the general public

Advantages can be listed as follows for each group:

(1) First and most importantly, certification will benefit EDP personnel since they should understand what standards must be satisfied to meet their basic job requirements. Thus, the individual would know what is expected of him. Used to its fullest, certification can be structured to form a career development program and, hence, be used to motivate and to give the employee a vocational goal. This approach requires a set of defined standards (tools of the trade) for

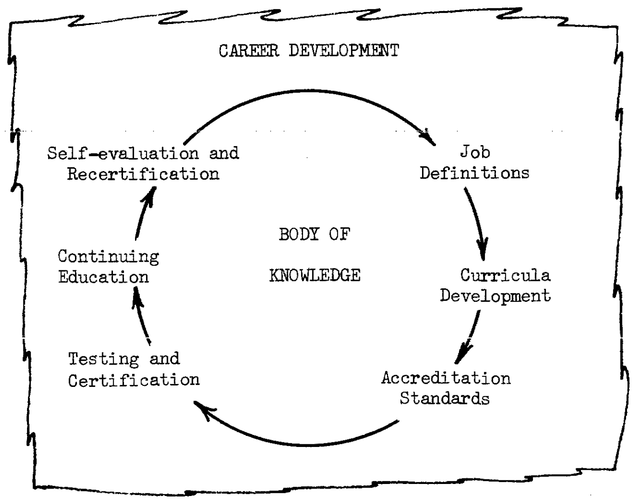


Figure 1—Certification cycle and career development[2]

each type of job and a matching job description. Finally, job performance criteria should be identified to each job and be used to measure the individual's performance.

(2) Certification would certainly help the employer when hiring or promoting computer personnel. If certification were done properly, the employer could rely more on the credentials of the individual. If the nature of the work required the person to be bonded or to handle sensitive information, the employer would have some assurance of the individual's competency if he were certified. Also, employers are no longer fooled by the mysteries of the computer. They have been *taken* too many times by poorly defined and developed software systems, and have come to expect the data processing function to pay its way. EDP budgets are being cut because of the failures of their Management Information Systems and poorly managed resources.

(3) The public sector should be protected against incompetency within the EDP ranks. The question which should be asked is: "Is the public health and safety, property rights or schooling of the young affected by the software and hardware of the computer community? If the answer is "yes," licensure should be required as discussed above. The best answer to this question is public opinion. Should EDP experts be required to attest in court to the validity of a well designed system or program? A license would most definitely be required in this instance.

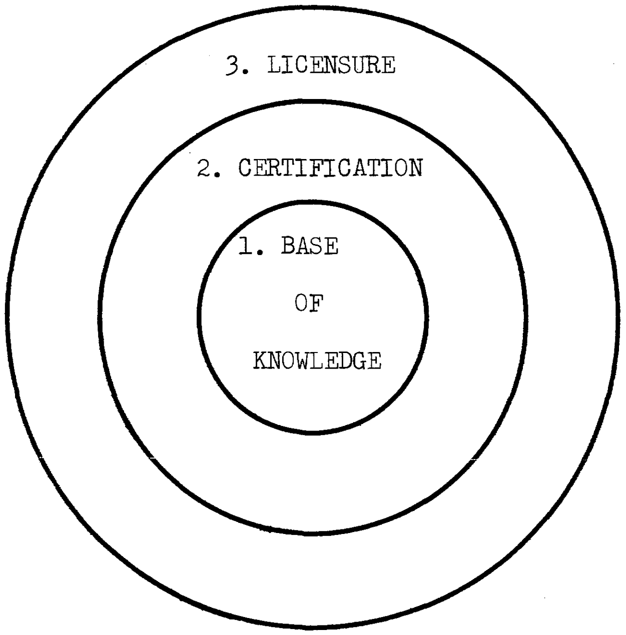


Figure 2—Steps to EDP professionalism

EFFORTS TOWARD EDP CERTIFICATION

As shown in Figure 1, the first step is to identify a body of knowledge including the definition of skills for each set of job descriptions. Next, such a body can be taught via curricular which can be controlled by accreditation standards. Once a body of knowledge has been identified and taught, testing and certification can be administered. Recertification can be given once a continuing education program has been identified. The cycle is reiterative since new jobs are continually being developed which require new or different skills.

JOB DEFINITIONS

For each job, skills must be identified and attached to each job description. Job descriptions should include:

1. reporting relationship of the position,
2. administrative responsibility,
3. technical or functional responsibility,
4. contacts made in the job,
5. supervision required in the job,
6. educational and experience qualifications,
7. specialized training and certification required

The first step toward job descriptions is to identify the job performance and to be able to measure such performance. Standard job descriptions have been identified by both the Civil Service Commission and the Association for Computer Programmers and Analysts (ACPA). The American Federation for Information Processing Societies (AFIPS) has been working on a set of job definitions and skill requirements, too. Much of this work has been performed by Dr. Ray Berger, director of Psychometrics, Inc. ACM has identified the skills necessary to meet the Information Analyst and Systems Designer job positions at the graduate and undergraduate levels.[3,4] These descriptions are examples of two such jobs which have been identified with specific skills. Work which has been done by these three groups would be good guidelines for the computer community to follow when identifying EDP job definitions.

CURRICULA DEVELOPMENT AND ACCREDITATION STANDARDS

Since standard job definitions can be identified with their associated skills, curricula can be developed to teach these skills. ACM has worked in this area with their Curricula for Information Analysts and Systems Designers. It has also done a large amount of work in Curricula Development for Computer Science programs which have been used throughout the United States. The University of Maryland, the American University, and the University of Minnesota, among others, have played a valuable role in implementing all or portions of the ACM Curricula for Information Analysts and Systems Designers. A number of universities and colleges have used the ACM Curricula for Computer Science as a guideline. Accreditation standards for such curricula in data processing have not been given the proper attention and should be devised. The AFIPS's report, "Professionalism in the Computer Field," states, "At the present time, there are no accrediting agencies specifically looking at Computer Science and Data Processing departments in the colleges and universities."[5] The article goes on to say, "The private EDP schools may seek accreditation by either the National Association of Trade and Technical Schools, the Accrediting Commission of Business Schools or the National Home Study Council. At the present time, a small minority of such schools are accredited by one of these three agencies."[5] Since job skills and job definitions have not been totally identified and, subsequently, standardized, little can be expected by the accreditation community to attest to the curricula being developed by individual EDP schools. Hence, such an effort is certainly needed by the EDP field.

TESTING

Testing can be grouped into two types, aptitude and proficiency. Aptitude testing verifies the ability of one to learn a certain base of knowledge. For example, if a computer operator wishes to become a programmer, his manager would like to know if the operator has the ability to undertake programming tasks. Another example is the inexperienced person who wishes to enter into the field of EDP. Several aptitude tests have been developed; the most popular one is the IBM Programmers Aptitude Test (PAT) which has been administered with some success.

The more difficult type of testing is proficiency. The reason this kind of test is difficult is that managers have not fully identified the level of performance for a particular type of EDP position. Also, standard skills and job definitions have not been fully identified to warrant ease of testing. The Berger Test of Programming Proficiency is one test designed to measure an individual's knowledge and proficiency in the basic principles and techniques of programming. It allows the individual organization to set its own standards of acceptability. Standard tests can be identified; Sweden has developed standard proficiency tests for most of its education courses throughout the country. Such a program should be researched and discussed as to the importance and feasibility of its application to EDP courses. Robert Dickmann has done extensive research and work in the area of testing. His book, "Personnel Implications for Business Data Processing," is a valuable piece of work in the area of testing EDP personnel.[6] Another area of proficiency testing is the one which verifies how well a person will perform on the job. This method of testing is very difficult because the personality of the individual is introduced and, hence, many variables come into the picture and are difficult to measure and define. Several companies are working toward developing tests in this area.

In his article, "Evaluating and Testing of Electronic Data Processing Personnel," Paul Oyer states that little has been done in the area of performance level testing, and standardized tests are needed to measure different levels of EDP achievement.[7] Some work has been performed which attempts to test or evaluate performance of systems analysts, programmers and operators. One such example is the performance test developed by Diebold Group Inc.; however, it has not been fully validated. Although this area of testing is only a portion of the certification procedure, it is vital to the effort and must be developed.

CERTIFICATION

Before August, 1973, only one organization, the Data Processing Management Association (DPMA), had become very involved with certification in the United States on a nation-wide scale. It sponsored two certification programs which were the Certificate in Data Processing (CDP) and the Registered Business Programmer (RBP) certificate. The CDP has proven itself as a tool for testing the entry level requirements of a data processing manager since its inception in 1962. It requires the applicant to answer 300 multiple choice questions in five general areas within 250 minutes. The areas of concentration are: data processing equipment, computer programming and software, principles of management, quantitative methods, and systems analysis and design. Over the past 10 years, about 14,000 EDP personnel have received the CDP certificate. The first CDP exam entailed only 100 questions. This certificate has slowly gained acceptance in many organizations such as Xerox, General Electric, Weyerhaeuser Co., Franklin Life Insurance Co. in Springfield, Illinois, State Farm Insurance, The Prudential Insurance Co. of America, and the U. S. Army Corps of Engineers . . . just to mention a few.

The RBP examination started its career in 1969 and attempts to test the skills of a business programmer. This test has questions which are multiple choice but has not been as popular as the CDP exam because of three major reasons: (1) data processing managers have not fully supported the exam, (2) the RBP has not been advertised as much as the CDP, and (3) the RBP is a later test and has not become as established as the CDP.

In August, 1973, the Institute for Certification of Computer Professionals (ICCP) became legally established as a non-stock corporation in the state of Delaware. It was formed primarily to certify, develop and recognize EDP personnel. Initially, the certification tools to be used are the CDP and RBP which are being transferred to the ICCP from DPMA. The ICCP was formed from what was originally titled the Computer Foundation. The following professional societies are members of the ICCP:

Association for Computing Machinery
Data Processing Management Association
IEEE Computer Society

Association of Educational Data Systems
Society of Certified Data Processors
Society of Professional Data Processors
Society of Data Educators
Canadian Information Processing Society
Association of Computer Programmers and Analysts
Automation/Association

If the certification movement is approached properly, this organization could have a great impact on the EDP community in addition to the general public.

Other certification programs which exist in the United States but are mainly related to areas other than EDP are the Certified Data Educator (CDE), Certified Internal Auditor (CIA) and the Certified Public Accountant (CPA). Since these are not considered to be directly oriented to the computer vocation, a detailed explanation of each will not be given.

Other countries have made impressive strides in certification. The two leading countries are Japan and England. Japan has a certification program which is equivalent to the CDP and RBP but consists of one test. There are two parts to its test, i.e., Class I and Class II. Class I is similar to the CDP and Class II is similar to the RBP. This program is administered by the Ministry of International Trade and Industry (MITI) and only 7 percent of the applicants in each class pass (as opposed to the 40 percent pass-rate in the U. S.). The questions of the tests are published for future study. (In the U. S., the CDP and RBP questions are not published.)

The British Computing Society (BCS) has a certification program which is by far the most extensive of any mentioned in this paper and probably the most extensive program in the world! Its primary purpose is to test those computer applicants who wish to join the BCS as a member. The exam involves two parts, and each part contains two sections. Sections 1 and 2 of Part I each require six hours of exams in two of the following six areas:

Part I. Section 1

1. Appreciation and development of computing systems.
2. Representation of data in the computer.
3. Set theory and the elements of logic design.
4. Appreciation of analog and hybrid computing systems.
5. Elementary programming.
6. Introduction to techniques for computer applications.

Part I. Section 2

1. Fundamentals of computer technology.
2. Programming.
3. Data Processing.
4. Analysis and design of information processing systems.
5. Computational methods.
6. Analog and hybrid techniques.

After the candidate has passed Part I or its equivalent, he can use one of two methods for Part II. One method is to take two more three hour examinations, plus a three hour written essay and an oral examination. The other method is to submit a dissertation on original work done at an advanced level. Part I was first given in 1969 and Part II in 1970. About 40 percent of the BCS candidates passed the Part I in 1972. Only eight passed the Part II requirements in 1972 from a group of 26 candidates. An interesting point is that the BCS has seven grades of membership, viz, Fellows, Members, Licentiates, Associates, Affiliates, Students and Institutional Affiliates. As an example, the grade of "Fellow" carries the following requirements: "... age over 30, and eight years accepted experience in data processing, five years of which must be in a responsible position. Exceptional merit over and above the normal call of professional duty must be proved."[8] Fellowship is (1) not normally considered until the applicant has been a member for at least one year and (2) granted only to members who can prove their professional activities justify acceptance as an authority in their particular field of data processing. The BCS exams are oriented toward scientific areas vice business data processing. One important decision which has been made by the BCS is the new ruling requiring that candidates applying for entry as members on the basis of experience and having had the required seven years must enroll as affiliate members. This must be done beginning December 31, 1973. Hence, those wishing to become members will eventually be required to take the examination because it has been recognized that years of experience do not necessarily equate to a certain level of knowledge. Finally, the BCS and Japanese exams have been governmently implemented, and centrally coordinated. The U. S. certification movement has not been centralized until the recent formation of the ICCP.

CONTINUING EDUCATION AND RECERTIFICATION

Most universities and colleges offer some kind of continuing education program. However, since no set of standard skills and job definitions have been proposed, it is difficult to structure a meaningful program for continuing education in EDP. Presently, there does not exist a recertification program in the U. S. or abroad.

CERTIFICATION AND CAREER DEVELOPMENT

One other factor which should be considered once the certification cycle has been identified and executed is a career development program such as that discussed by Nancy Ayer.[9] At this juncture of the certification problem, the primary goal should be oriented toward the EDP individual vice licensure. Figure 3 depicts an example of a proposed career structure with its career ladders. (For more detail on this structure, refer to the article, "Step-by-Step: A Career Structure for Systematic EDP Growth.")[10] Such a program can be used to measure progress toward certification of an EDP individual within a company. In fact, if

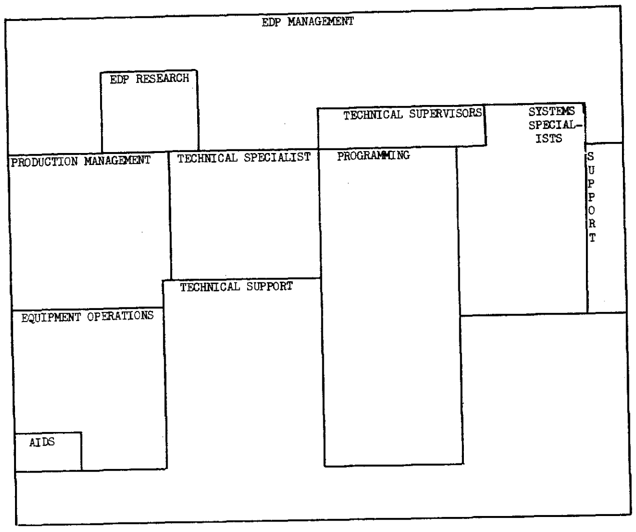


Figure 3—EDP families within a career development structure[10]

such a program could be identified as a standard guideline, it would lend itself to helping the individual in planning his career and would provide a means for continuing his education and testing his on-the-job progress. Such a program is being identified by the ACM Special Interest Group of the Washington, D. C. Chapter, chaired by Paul Oyer of the Census Bureau. The career development structure being proposed contains 10 different groups of similar jobs, called "Families." An example of a family is the computer operator series which in turn is composed of four levels of operators, i.e., Computer Operator I, II, III, and IV. Each level has a job description with specific skills identified. Further, each level has associated with it the performance expected from the employee. Progress of each employee could be measured by testing, job performance and the supervisor's recommendations. Each level has a set of training courses identified to it, both those required and those that are optional. Once the employee has satisfied the requirements of his job level and has demonstrated his potential for advancement, he has two primary options available to him: (1) to continue in the same family at a higher level, or (2) to change to another family if an entry point exists. Such a career development structure requires five types of tests: (1) entry into a basic family for the first time, (2) entry into a higher level job within the same family, (3) performance and subject testing at the same level, (4) entry into a higher-level family from another organization, and (5) entry into a different family (e.g., operator to programmer). Such a career structure would lend itself to assisting the certification effort. This structure contains all the essential parts of certification as mentioned previously. It utilizes testing, personnel references, job performance and experience, communications, EDP contribution, and education. Another important point which this structure shows is that certification programs should consist of more than one or two types of certification. In this example, ten different types of certificates could be awarded, assuming each required certification. Entry level for an inexperienced computer person would not be involved in certifying an individual since he has no experience. In this case, aptitude of the individual should be tested. However, a career development program should not ignore the importance of screening those individuals who are not well adapted to the EDP field. If all areas are considered and included in a certification program, the career development framework would greatly benefit the individual, the employer and the general public. Greater success of implementing systems would be realized, thus reducing costs and personnel turnover.

FUTURE OF EDP CERTIFICATION

It is very difficult to give justice to such an all encompassing subject as Certification in such a short space. What has been given is a general view of the status of EDP certification and some ideas to be considered. The first step to take in any situation is to recognize the problem. The formation of the ICCP is the first important step to recognize the plight of the EDP community. The next step is to determine exactly where we are and what efforts have been taken to remedy the problem. Work is being done by the SIGCD of Washington, D. C. which will aid the computer community in this effort.[11] Some very significant work has been done in the areas of job definition, particularly by AFIPS, ACM and ACPA. EDP curricula have been defined but its total validity is questionable since no standard has been formulated for job skills or even job types. Accreditation standards must become an important factor in certification. This can be done with the assistance of EDP societies, or more appropriately, with the assistance of ICCP. Much work has been done toward testing EDP personnel; however, job performance in the present job or in the next higher job has not been validated. More effort should be oriented toward the testing of oral and written capabilities of the individual. One aspect which the CDP and RBP tests do not attempt to do is to verify the oral and written competency of the applicant. Essay questions should be given for certain "families" or for all "Professionally" identified groups. If certain levels of competency are identified as in the BCS, certain levels of certification should be allowed. Possibly, the highest level of a certain professional family should be required to (1) do original work which will contribute to the EDP profession and (2) pass an oral exam on this original work. The written essay exam would prove the candidate's ability to write and to logically approach a problem. Many universities and colleges require one or more comprehensive exams at the masters level and some even require such exams at the undergraduate level. Testing criteria could be identified to grade the oral and written exams. The CDP is certainly not the answer to EDP certification since there can be several different groups or "families" in the EDP field. The CDP has proven to be very valuable when considering such aspects as administering a nation-wide test, verification of EDP questions, etc. Continuing education must be formulated to support the recertification step. Little has been done on identifying such a program. EDP personnel must be properly educated.

Once the problem has been identified and the present situation has been inventoried, the next major step is to set forth objectives and requirements. We must define "Professional" EDP areas and determine if certification is required before the individual can practice. The present status of EDP certification is such that licensure should not be considered until a framework with definitions has been built. The first priority to be satisfied is to give direction to the individual and, subsequently, to the computer vocation. Once this has been done, licensure and ethics can be confronted. The first priority *must be met* by the EDP universe or the public will place demands upon the EDP community through state or federal legislation. Such a move would be premature since we have not defined or set standards.

The fourth step involves the formulation of an approach to achieve certification. Hopefully, this will revolve around the individual and will entail a centralized effort to establish a path toward a meaningful certification program. Also, of great importance is the forming of a career development structure which will enhance EDP certification. If a career development program is implemented, it will aid EDP personnel in keeping abreast of their area of expertise. It will motivate them to strive for the next higher level of certification. In the author's opinion, certification exams should not require the applicant to have a degree for two reasons: (1) people who have certain undergraduate degrees do not always out-perform those without a degree, and (2) experience if of a progressively concentrated nature should be accepted in lieu of certain types of formal schooling. Everyone should be required to pass their level of certification via the successful completion of an examination. If the programmer is considered a "Professional," the certification exam should be difficult enough that a college degree in a certain area of endeavor would greatly benefit the individual in passing the exam. Recertification is urgently needed. Because of the dynamics of EDP, an individual can quickly become obsolete. Any formulated certification program should address itself to this aspect. In order to keep a certificate current, EDP personnel should be required to take a "mini" exam to verify that the candidate is up-to-date in the field. Continuing education should support the recertification program and is necessary throughout the career of each person. Such areas as technology changes, management concepts, computer metrics, software enhancements, terminology, etc., must be covered in such a program. Licensure should not be imposed in the near future because the EDP community is not prepared. Even a base of knowledge has not yet been standardized.

Hopefully, the ICCP will be the answer to the EDP certification problem by defining all certification steps in detail and not just the testing aspect. Much coordination and research work remains to be done; if undertaken properly, a successful EDP certification program can be implemented. If the objective of the certification movement is oriented toward the individual, the answer is "yes" to the question, "EDP Certification—Is It Necessary?" If the objective is to license the EDP individual in the near future, the answer is "no."

In the late 50's and early 60's, the primary concern of the computer world was the sophistication and gadgetry of the computer hardware. Next followed the concern for the state-of-the-art in software, its capabilities, structure and ease of use. However, the 70's are pointing in the direction of computer peopleware and the ability of top management to ascertain the quality of these people. The thrills of hardware and software must come second to the needs of the EDP individual. The decade of the 70's should best be remembered as the "peopleware" era and not the "fourth generation" era.

REFERENCES

1. Harris, Fred H. and John K. Swearingen, "Report on the Status of the Institute for Certification of Computer Professionals," *Data Management*, October, 1973, p. 18.
2. Ibid.
3. Teichroew, Daniel, Ed. "Education Related to the Use of Computers in Organizations," *Communications of the ACM*, September, 1971, pp. 573-588.
4. Couger, J. Daniel, Ed. "Curriculum Recommendations for Undergraduate Programs in Information Systems," *Communications of the ACM*, December, 1973, pp. 727-749.
5. *Professionalism in the Computer Field*, The American Federation of Information Processing Societies, Montvale, New Jersey, 1970, p. 7.
6. Dickmann, Robert A., *Personnel Implications for Business Data Processing*, John Wiley and Sons, Inc., New York, New York 1971.
7. Oyer, Paul, "Evaluating and Testing of Electronic Data Processing Personnel," (Unpublished paper). Available from the Institute for Science and Public Affairs, Box 9242, Suitland, Md. 20023.
8. "Experience Only! Door is Closing," *Computing*, July 6, 1973, p. 20.
9. Ayer, Nancy, "Step-by-Step: A Career Structure for Systematic EDP Growth," *NCC '74 Proceedings*, Volume 43.
10. Ibid.
11. Ibid.
12. Oyer, Paul and Dorothy L. Ray, "A Systems Approach to Career Development: Report of Two surveys," *NCC '74 Proceedings*, Volume 43.

BIBLIOGRAPHY

Ayer, Nancy, "Step-by-Step: A Career Structure for Systematic EDP Growth," *NCC '74 Proceedings*, Volume 43.

Beyer, Jack, "The Society of Data Educators," *Data Management*, November, 1972, p. 33.

Canning, Richard, "The Question of Professionalism," *EDP Analyzer*, December, 1968, p. 1-1.

———, "Career Programs in Data Processing," EDP Analyzer, August, 1971, pp. 1-15.

Couger, J. Daniel, Ed., "Curriculum Recommendations for Undergraduate Programs in Information Systems," *Communications of the ACM*, December, 1973, pp. 727-749.

"Curriculum 68," *Communications of the ACM*, March, 1968, pp. 151-197.

Dickmann, Robert A., *Personnel Implications for Business Data Processing*, John Wiley and Sons, Inc., New York, New York, 1971.

Dorn, Philip H., "EDP Professionals—The Blurred Image," *Datamation*, Jan., 1971, pp. 22-24.

"Experience Only! Door is Closing," *Computing*, July 6, 1973, p. 20.

Glaser, George, "AFIPS—An Inside View," *Data Management*, November, 1973, pp. 30-33.

Gilchrist, Bruce and Milton R. Wessel, *Government Regulation of the Computer Industry*, AFIPS Press, Montvale, New Jersey, 1972.

Guerrieri, John A., Jr., "Certification—Evolution, Not Revolution," *Datamation*, November, 1973, pp. 101-104.

Harris, Fred H. and John K. Swearingen, "Report on the Status of the Institute for Certification of Computer Professionals," *Data Management*, Oct., 1973, pp. 18-21.

Inoue, Michael S., Ph.D., "Japanese CDP Program—Number Two Tries Harder," *Data Management*, February, 1971, pp. 26-31.

Oyer, Paul, "Evaluating and Testing of Electronic Data Processing Personnel." Unpublished paper, (Available from the Institute for Science and Public Affairs, Box 9242, Suitland, Md. 20023).

Oyer, Paul and Dorothy L. Ray, "A Systems Approach to Career Development: Report of Two Surveys," *NCC '74 Proceedings*, Volume 43.

Professionalism in the Computer Field, The American Federation of Information Processing Societies, Montvale, New Jersey, 1970.

Reinstedt, Robert N. and Raymond M. Berger, "Certification: A Suggested Approach to Acceptance," *Datamation*, November, 1973. pp. 97-100.

"Should Computer Professionals Be Licensed?" *Communications of the ACM*, May, 1971, p. 368.

Skeen, David R., "ADP Professionalism . . . A Myth or a Reality?" unpublished paper. (Can be obtained upon request from the Author).

Teichroew, Daniel, Ed., "Education Related to the Use of Computers in Organizations," *Communications of the ACM*, September, 1971, pp. 573-588.

Skills possessed and skills useful for MIS practitioners—A research report*

by ROBERT M. HENRY

University of Minnesota
Minneapolis, Minnesota

The major report setting forth major curricular recommendations for graduate professional programs in information systems appeared in 1972.** The context of the report was an information systems environment and developmental process distinguishing between two analyst activities: information analysis and design analysis. Figure 1 seeks to draw the ACM distinction. And that distinction is carried forth into skill sets identified as desirable for graduates. Six clusters or groupings of skills are offered (people, models, systems, computers, organizations, and society) and thirteen courses are proposed to impart those skills. The thirteen courses are set forth as a two-year graduate core curriculum for *both* information and design analysts.

The Management Information Systems Research Center at the University of Minnesota has taken the liberty of amending the chart (Figure 2) from its form in the ACM report. Course titles and sequencing are unchanged. The rows in the chart represent the four semesters of core courses in the recommended two-year curriculum. Vertical or downward sloping arrows indicate prerequisite relationships. Horizontal arrows represent co-requisite course offerings. The chart shows the names of skill clusters identified by the committee nearest the courses which most directly address those clusters. Thus "people" skills are seen to be communicated through courses entitled "Organizational Functions" and "Human and Organizational Behavior." At the bottom of the exhibit, the Management Information Systems Research Center has added the words "Information Analyst" and "Design Analyst", each with an arrow. The arrow indicates the opposing poles of the curriculum at which either analyst might take additional elective courses. Six of the courses are shown to be more representative of generalist information analyst skills and seven of the courses are seen to be more representative of the design analyst's specialty area. The words "generalist" and "specialist" are MISRC additions.

The ACM report isolated six skill clusters in proposing thirteen graduate courses spread over two years. While opinion was solicited by the ACM committee from many areas, no formal research underlay the ACM report. The University of Minnesota sought to validate the ACM assumption and recommendations. The ACM committee assumed sufficient demand for Information Systems graduate students to justify their proposed curriculum and recommended specific skills to be stressed by that proposed curriculum. To test the ACM assumption and recommendations, the MISRC researchers determined to survey DP/IS practitioners in an empirical fashion. Relevant questions included the following: What skills do practitioners presently possess? What skills are useful for each position? How do employees, supervisors and users vary in their perceptions? What then are the implications for education?

The MISRC research objectives were three in number:

1. Survey projected demand quantity for MIS graduates
2. Specify skills required
3. Develop curricular implications

The procedure in attacking these research objectives was first to build a subject sample. Seventeen relatively autonomous organizations were approached in fourteen Twin Cities firms. In concert with the Corporate Director of Information Systems or his equivalent, a manning table was developed for twelve positions representing the managerial and software development positional levels within information system. A descriptive paragraph attempted to set the bounds for each position so that, regardless of job titles, like functions and personnel could be compared across organizational boundaries.

In the next step skills were selected, clustered, and processed into research instruments. After pilot testing, 111 skills were eventually selected to be researched. The skills were grouped into seven clusters and then reclustered into three. The total skill set appears in Appendix A. An asterisk

* Sustaining MISRC Associates for the research period included Burlington Northern, Inc.; Dayton Hudson Corporation; CENEX, Inc.; Federal Reserve Bank; General Mills, Inc;. Honeywell, Inc.; International Multifoods; Minneapolis Gas Company; 3M Company; Northern States Power Company; Northwest Bancorporation; Pillsbury Company; St. Paul Companies, Inc.; and Soo Line Railroad Company.

** ACM Curriculum Committee on Computer Education for Management. "Curriculum Recommendations for Graduate Professional Programs in Information Systems." *Communications the ACM,* 15:5, May, 1972.

Information Analysis	Activity	Design Analysis
Organizational Dynamics	Focus	Computer Dynamics
Determination of Information Needs	Task	Translation into Hardware/Software
(1) Feasibility (2) System Specification	Phases	(1) System Design (2) System Implementation
Product Design	Analogy	Manufacturing System Design

Figure 1

DISTRIBUTION OF GENERALIST AND SPECIALIST SKILLS WITHIN SIX ACM CLUSTERS

	Generalist Cluster		Specialist Cluster		Performance Cluster
People	10		0		
Organizations	12		0		
Society	10		0		
Systems	11		12		
Models	1		9		
Computers	0		32		—
	44	+	53	+	14 = 111

Figure 3

on any skill in the first six clusters represents a "generalist" skill or one more representative of the information analyst activity. Skills without asterisks in the first six clusters represent "specialist" skills more appropriate to the design analyst activity. The seventh cluster, "performance", cannot be divided into specialist or generalist skills.

The performance cluster is a part of each clustering scheme (see Figure 3), illustrated between the six rows and three columns.

Once the firms were selected and the skill sets were chosen, pilot tested, and clustered, separate instruments were designed for supervisors, users, and employees. Each participant supervisor and user rated a designated employee of his acquaintance at a level within the organization determined by the researchers and the information systems manager. Supervisors and users rated the employees as to skills actually possessed and skills deemed useful or non-useful for that employee's level of functioning, regardless of whether or not the employee actually possessed the skill. Rating scales were a four-piont forced choice. A pilot attempt was begun with six points rather than four, but the four-point scale was found better to harmonize with four-point scales presently being used by subjects in job review and evaluation. For every one of the 111 skills, then, supervisors and users rated a given employee as to the degree of skill possessed and the usefulness of that skill for the position. Where a rater was not familiar with one of the 111 skills, where a rater had not observed either the presence or absence of that skill in the employee being rated, or where the rater had no feel for the usefulness of that skill for the positional level within the organization, the rater was requested to refrain from making a judgment.

Each employee engaged in self rating as to skill levels possessed. Just as the supervisors and users, they rated the usefulness or non-usefulness of every skill for their positional level in the organization. Unlike supervisors, employees were asked to designate the source (or sources) of skills for each skill they possessed. Three weights could be assigned by employees as to the sources for any skill. In the illustration, shown by Figure 4, the employee has indicated he is superior or exceptional in his ability to write detailed program specifications. That skill is deemed by him to be extremely useful or significant for his position in the organization. The employee has indicated that primary sources of that skill for him were: (1) higher education; (2) on the job experience after data processing entry; and (3) in-house education. Firms were chosen, skills selected, and rating instruments designed. Data gathering was step four. The subject sample consisted of 981 persons, representing 475 employee raters, 375 supervisor raters, and 131 user raters distributed among twelve positional levels. (See Figure 5).

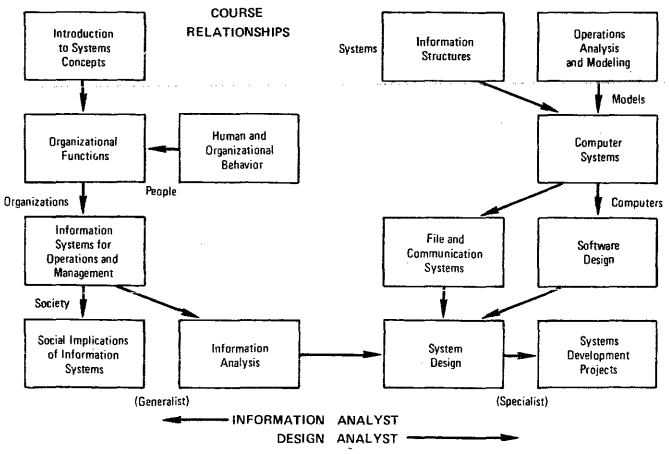

COURSE RELATIONSHIPS
Introduction to Systems Concepts
Organizational Functions
Human and Organizational Behavior
People
Organizations
Information Systems for Operations and Management
Society
Social Implications of Information Systems
Information Analysis
Systems
Information Structures
Operations Analysis and Modeling
Models
Computer Systems
Computers
File and Communication Systems
Software Design
System Design
Systems Development Projects
(Generalist)
(Specialist)
INFORMATION ANALYST
DESIGN ANALYST

Figure 2

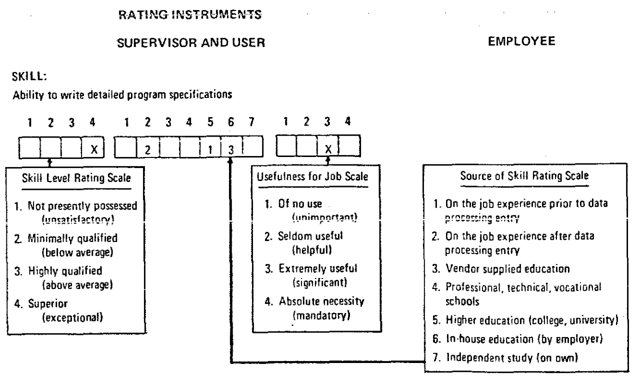

RATING INSTRUMENTS
SUPERVISOR AND USER
EMPLOYEE
SKILL:
Ability to write detailed program specifications
1 2 3 4
X
1 2 3 4 5 6 7
2 1 3
1 2 3 4
X
Skill Level Rating Scale
1. Not presently possessed (unsatisfactory)
2. Minimally qualified (below average)
3. Highly qualified (above average)
4. Superior (exceptional)
Usefulness for Job Scale
1. Of no use (unimportant)
2. Seldom useful (helpful)
3. Extremely useful (significant)
4. Absolute necessity (mandatory)
Source of Skill Rating Scale
1. On the job experience prior to data processing entry
2. On the job experience after data processing entry
3. Vendor supplied education
4. Professional, technical, vocational schools
5. Higher education (college, university)
6. In-house education (by employer)
7. Independent study (on own)

Figure 4

N = 981	Participants Emp	Participants Supvr	Participants User
Director/Manager of all D.P.	12	7	2
Assistant Manager of D.P. or Specialty Manager of D.P.	29	25	9
Manager of Systems Analysis	18	17	10
Project Team Director (Lead Analyst/Programmer, etc.)	30	25	8
Senior Systems Analyst	48	42	18
Junior Systems Analyst	25	19	2
Manager of Analysis and Programming	31	26	12
Senior Systems Analyst/Programmer	84	67	34
Junior Systems Analyst/Programmer	50	42	15
Manager of Programming	4	2	1
Senior Programmer	84	53	13
Junior Programmer	60	50	7
TOTALS	475	375	131

RESEARCH SAMPLE
Figure 5

EMPLOYEE GRADUATION DATA

	N	%
High School	89	18
Technical Institute	83	17
Undergraduate Degree	250	53
Graduate Degree	34	7
Undergraduate and Technical Institute	17	4
Graduate and Technical Institute	2	1
	475	100

Figure 7

The data base consisted of nine major files—two files each for employees, supervisors and users consisting of their 111 ratings both for (1) Skills Possessed and (2) Skills Useful. There were three additional employee files: (1) company demographic information (size of firm, monthly hardware expenditure, reporting structure, etc.); (2) individual demographic information (total DP experience, length of time per present position, immediate past ten-year work history, educational level, graduation data, major specialty area, age, etc.); and (3) skill sources for the 111 skills. Output at any data base entry includes (1) skill mean ratings, (2) skill sources, (3) a rank ordering of the entire skill set, and (4) individual and company demographic data. Inquiries may be addressed for any or all positions by any set or sub-set of either rated or rater subjects, for skills possessed and for skills useful, for each of the 111 skills and/or each of the skill clusters. T-test significance testing was at the .001 level.

MAJOR FIELD IN COLLEGE

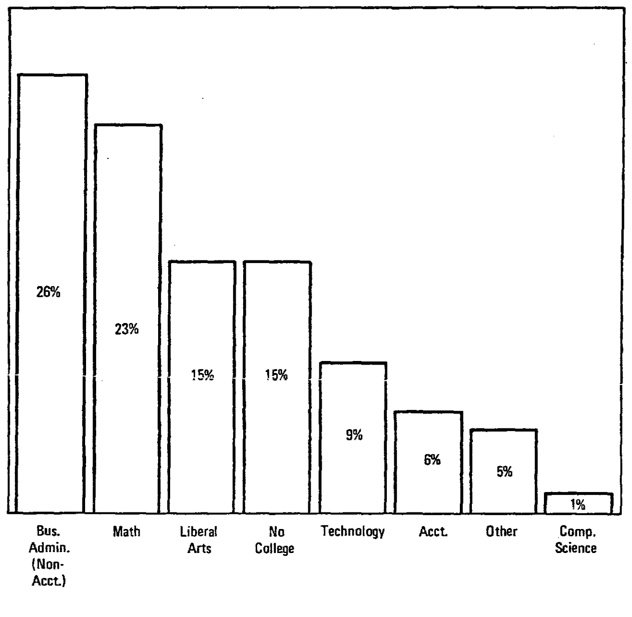


Figure 6

FINDINGS

Company demographic information

The firms have major hardware expenditures for DP/IS averaging slightly more than $75,000 per month for mainframes in Twin Cities locations alone. In all but five cases the director's reporting channel is through the Vice President for Administration or Management Services (or his equivalent). Analysts and programmers are assigned about equally to applications areas and to project teams. The user's role has become increasingly prominent. Essentially, they represent information systems as opposed to DP environments.

Individual demographic information

Subjects averaged 30 years 11 months in age, six years mean total DP experience plus an additional one and one-half years non DP-business experience. Average DP experience per current firm was slightly more than five years and the average length of time per present position was two and

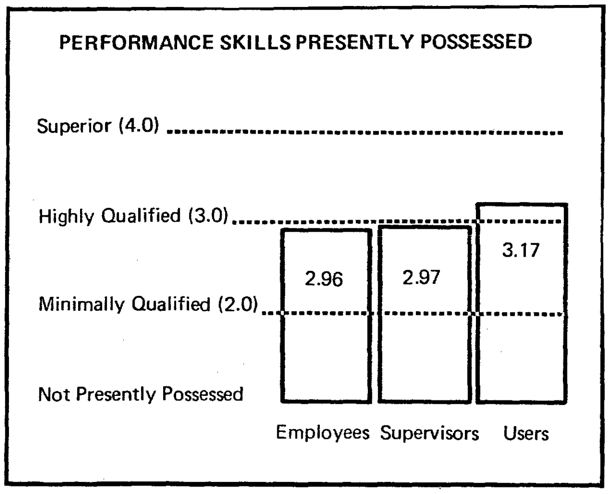


Figure 8

one-third years. Number of companies averaged 1.79 to include present employer. All demographic figures were slightly deflated due to two factors: (1) work histories were reported only for 120 months; and (2) the sample was biased toward newer hires and promotees.

Only 15 percent of the sample had not attended college. Majors or specialty areas are shown in Figure 6. As these are combined, the subjects were one-third Business Administration, one-third Math and Technology, one-sixth Liberal Arts, and one-sixth no college.

Figure 7 is the graduation data for all subjects. Sixty-five percent of the sample had college or university undergraduate degrees. Eight percent earned graduate degrees.

Skill cluster means

By every comparison of the performance cluster with every other cluster, for any position, by all raters, on both skills possessed and skills useful, the performance cluster received a significantly higher mean rating. There was no surprise in terms of usefulness. The big surprise was that users (1) rated employees significantly higher than either employees or supervisors on performance skills presently possessed (Figure 8); and (2) rated performance skills significantly higher than any other cluster of skills presently possessed (Figure 9).

In Figure 9, the rank ordering of employee and supervisor mean ratings were consistent for levels of skills possessed and relative usefulness by cluster in both the seven cluster and the three cluster analysis. Users disagreed little as regards relative usefulness, reversing only the importance of the society and models clusters in the sixth and seventh rank order positions for the seven cluster comparison. Users disagreed drastically on the cluster mean values derived from their individual ratings on skills presently possessed. After surprising by elevating the performance cluster, users on both the seven cluster and three cluster groupings said that the twelve positions surveyed possess technical skills above all else. In short, the users confirmed the projected image of "computerniks" on the DP/IS personnel. And that image is the more strongly stated in contrast with their clear agreement with employees and supervisors as to rank orderings of useful skills.

The rank ordering varied but not substantially across positions. Specialist skills would be elevated, for example, over generalist skills both for senior and junior programmers. At all other positions generalist skills were deemed more useful. The marked trend to elevate the behavioral skills with systems over technical skills suggests something of the evolution of systems, the enhanced role of the user, and the viability of the ACM insight in suggesting an information analyst as a key developmental resource.

Entry level positions

The probability of entering each of the twelve positions is depicted in Figure 10. From "without" means either from outside the DP/IS organization or from outside the firm. Experience to acquire threshold organizational and specialty skills is predictably more determinative than graduate versus undergraduate levels of information analysis or design analysis education. With good DP/IS experience of 1-3 years, either design or information analysts may target for junior systems analyst/programmer or junior analyst, depending upon the job function within the organization. Additional experience will enhance opportunity for more rapid upward mobility. Rarely will graduates enter at senior levels of MIS environments until tested within the environment. Graduates recruited who have not yet acquired DP/IS work experience will enter, at least

RANK ORDERINGS OF SEVEN AND THREE SKILL CLUSTERS
BASED ON CLUSTER MEAN RATINGS

Rank Order Position	Skills Possessed		Skills Useful	
	Emp & Supvr Raters	User Raters	Emp & Supvr	User
1	Performance	Performance	Performance	Performance
2	People	Computers	People	People
3	Systems	Systems	Systems	Systems
4	Organizations	People	Organizations	Organizations
5	Computers	Organizations	Computers	Computers
6	Society	Society	Society	Models
7	Models	Models	Models	Society
			Emp & Supvr & Users	
1	Performance	Performance	Performance	
2	Generalist	Specialist	Generalist	
3	Specialist	Generalist	Specialst	

Figure 9

presently, at the junior programmer or DP trainee level (regardless of degree level). If the programmer function continues to shift toward a coding function, entry level for the environments surveyed will be standardized at junior analyst or junior programmer analyst level. While the openings were not surveyed in the present research, the evaluation of MIS oriented systems suggests that increasingly information and design analysts with strong functional competence in an applications area, *e.g.*, finance, marketing, etc., will have recruitment opportunity from the applications area of the firm. Although experience is key, the desired combination of experience and relevant education has implications for both recruitment areas—the firm and the educational institution. As the firm must select "mid-career" persons for additional education, so must higher education recruit them.

Demand

Of the environments surveyed an average of three new junior programmer lines, two new junior analyst/programmers, and one new junior analyst is projected each year for each of the next five. Private talk indicates the forecasted increase is at asking levels only. The curve of the increase for both senior and junior programmers is decreasing. The curve of the increase for both junior analysts and junior programmer/analysts is increasing. The net size of the organization has essentially stabilized. Information rather than design analysts appear the critical new resource required.

Sources of skill

Figure 11 shows the percentage of each of seven sources as contributor to a given skill cluster. The same analysis

<u>Probability of Entry from Without</u>

1.	Director/Manager of All Data Processing	6%
2.	Assistant Manager of Data Processing or Specialty Manager of D.P.	5%
3.	Manager or Supervisor of Systems Analysis	4%
4.	Project Team Director (Lead Analyst/Programmer, etc.)	7%
5.	Senior Systems Analyst	15%
6.	Junior Systems Analyst	15%
7.	Manager of Analysis and Programming	5%
8.	Senior Systems Analyst/Programmer	17%
9.	Junior Systems Analyst/Programmer	35%
10.	Manager or Supervisor of Programming	0%
11.	Senior Programmer	25%
12.	Junior Programmer	100%

Figure 10

SKILL CLUSTER SOURCES BY PERCENTAGE
ALL EMPLOYEES

Cluster	SKILL SOURCE						
	On The Job Training Prior to DP Entry	On The Job Training After DP Entry	Vendor Supplied Education	Technical Institute	Higher Education	In-House Education	Independent Study
People	18	40	2	1	18	5	15
Systems	5	53	8	4	12	9	10
Computers	1	49	15	4	7	12	11
Organizations	10	51	4	3	16	8	8
Society	5	45	4	2	14	4	25
Models	3	30	7	3	41	6	10
Performance	19	43	2	2	15	5	13
Generalist	18	25	5	3	22	9	18
Specialist	3	30	18	6	14	16	14

Figure 11

can be performed on each of the 111 skills, where possessed. The informal process of "on the job training after DP entry" was the major source cited. Higher education was the highest formal process cited, somewhat surprisingly given the mean seven and one-half years subject aggregate business experience and the newness of computer-related curricula. The source of tomorrow's information analyst is predictably higher education.

Continuing education

Space does not permit the position-by-position profile which must be the address of another paper. On the basis of supervisor usefulness ratings for the ACM clusters significant continuing education needs by position were as follows:

1. People—all positions except senior and junior programmers
2. Systems—all positions except specialty managers, senior and junior programmers
3. Organizations—all positions except specialty managers, programming manager, senior and junior programmers
4. Computers—throughout the programmer/analyst and programmer functions
5. Society—for the director, junior analyst, and programming manager
6. Models—for the assistant director and the specialty manager

SUMMARY

A large data bank has been constructed which may be queried on numerous parameters. Only gross findings have been reported to date.

Certain ACM assumptions and recommendations were tested. For the environments sampled, there is viability and greater demand for information analysts than for design analysts. If graduates could undergo the two-year recommended curriculum "mid-career" with 1-3 years of specialty experience, a highly recruitable graduate could be produced. Experience will increase as a dominant recruitment criterion.

An interesting consideration for further research is to compare MISRC's 1973 research profile with another for the same twelve positions obtained in 1975. Updated implications for education might again be produced.

APPENDIX A—GENERAL INDEX TO SKILL CLUSTERS

• Generalist or Information Analyst
* Specialist or Design Analyst

NOTE: All skills are preceded by the words "Ability to" or "Knowledge of."

I. PEOPLE (10)

* • communicate with others verbally (in general)
* • describe and identify individual and group behavior (e.g., describe and identify working relationships among people in an organizational environment)
* • predict alternate future behavior of individuals and groups (e.g., predict individuals' reactions to operating changes)
* • grasp the facts and feelings of what is spoken
* • recognize, understand, and communicate the meaning a particular event has for you
* • interview others
* • effect change in work relationships
* • communicate and interact with *non-computer* oriented people
* • gain the confidence and support of others in work relationships
* • recognize and remove personality problems which interfere with job completion

II. SYSTEMS (23)

* • view, describe, and define any situation as a system
* • analyze and evaluate different software *applications* packages
* • perform economic analyses (cost/benefit studies) of proposed resource commitments for a project
* • analyze and determine cost benefits of project (information system) to user
• design and use I/O layouts
* • calculate cost/performance tradeoffs in a system
* • present in writing a detailed description of part of a project
* • evaluate system performance and make needed adjustments to system after implementation
• design and use decision tables
• design and use run and grid charts
* • specify, given information needs and sources, several alternative sets of information to meet needs
• develop the major alternatives in specifying an information processing system, including data files and communications structures
• prepare clear and useful documentation (programs and procedures within programs, systems, etc.)
• analyze programs outlines by system analysts for detailed design and construction
* • make "rough cut" feasibility evaluations of proposed new techniques or applications of current technology
• design software and hardware configurations
• design and use flowcharts (system and program)
• prepare effective user documentation for either a portion of a system or an entire system
• project planning and control tools (PERT, CPM, etc.)
* • sources for updating knowledge of technology
* • general systems theory (open/closed systems, system boundaries, feedback concept)
the need for security in programs, data storage, and work flow design, as well as physical protection of programs and data
"outside" computer services (information concerning consultants, software houses, application packages, etc.)

III. COMPUTERS (32)

• write detailed program specifications
• convert existing programs from one system to another (language to language, computer to computer)
• use program testing aids (special debugging packages, traces, and snapshots)
• program in file oriented languages (COBOL, RPG)
• program in scientific or algorithmic type languages (FORTRAN, PL/1)
• program in an assembly type language (BAL, COMPASS)
• program in simulation languages (GPSS, SIMULA)
• revise existing programs (including debugging and refinement)
• analyze communication systems (estimate line and terminal requirements, volume and message length, queues, etc.)
• use sort and utility packages
• use sequential and index sequential file techniques
• use direct or random file techniques
• create, maintain and interrogate files
• prepare sample data for programs and providing test runs
• analyze and evaluate programming languages for selecting most appropriate language for a given problem solution
• analyze and evaluate different hardware configurations
• use interactive debugging facilities (available through a time-sharing arrangement such as Text-Editor)
• existing communications facilities (line types, exchanges, utilities)
• multilinked data structures (trees, multilist, inverted list, networks, etc.)
• sorting techniques (radix, merge, bubble, tree)

- searching techniques (sequential, binary, directory)
- microprogramming
- performance evaluation techniques (simulation packages, hardware and software monitors)
- minicomputers
- characteristics of auxiliary storage devices (capacity access, storage): tape, disk, drum, etc.
- input-output devices (types available, general market)
- operating systems (including scheduling algorithms, memory and peripheral management, interrupt systems)
- multiprogramming and multiprocessing
- time-sharing operating system (concepts and facilities)
- job control languages (coding and techniques)
- "inner workings" of compilers, interpreters or other translators
- communication access methods and their general features to support terminal/teleprocessing applications

IV. ORGANIZATIONS (12)

*• develop positive and negative impacts of a specified information system on specified parts of an organization
*• identify in an on-going organizational situation the key issues and problems of a given functional area (production, finance, marketing, etc.)
*• present in writing a summary of a project for management action (suitable to serve as a basis for decision)
*• identify possible short term and long term effects of a specified action on organizational goals
*• develop specifications for a major information system, addressing a given organizational need, and determine the breakdown into manual and computer-based parts
*• gather information systematically within an organization, given specified information needs and/or specified information flows
*• apply the "system viewpoint" in depth within the organization structure
*• data gathering techniques (interviews, etc.)
*• the function of purposeful organizational structure and the major alternatives for that structure
*• specify elements and relationships of information in various functional segments of the organization
*• accounting practices and procedures
*• corporate policy and lines of authority and responsibility

V. SOCIETY (10)

*• articulate and defend a personal position on some important issue of the impact of information technology and systems on society (important as defined by Congressional interest, public press, semitechnical press, etc.)
*• public and private data banks
*• computer impacts on industrial, clerical and managerial positions
*• computer industry with regard to growth patterns, competition, and government regulations
*• standardization practices in the computer industry
*• professional data processing associations
*• problems of providing training in data processing
*• changes in employment patterns as a result of automation
*• potential applications of automated processes for society
*• evaluate the social consequences of a proposed system

VI. MODELS (10)

- formulate and solve simple management science type models, linear programming, dynamic programming, queueing, simulation)

*• recognize the appropriate management science (operations research) models for situations commonly encountered

- queueing structures
- inventory control models
- matrix algebra
- differential calculus for optimization
- elementary statistics
- fundamentals of probability theory
- set theory
- formulate and solve complex simulation models

VII. PERFORMANCE CLUSTER (14)

- accept responsibility and initiate action
- perform tasks accurately
- cooperate and work effectively with others
- plan and organize work assignments
- motivate self and others
- train and develop subordinates
- create in others an acceptance and willingness to discuss problems
- manage other people
- complete assignments on time
- work effectively under pressure
- work independently with limited supervision
- define a problem
- handle a number of assignments simultaneously
- delegate assignments and review the results of assignments directly under control

Data base—An emerging organizational function

by RICHARD L. NOLAN

Harvard University
Boston, Massachusetts

A vignette of increasing frequency is for a manager with a problem to arrive in the EDP manager's office and ask for information required to solve his problem. The information required almost always cuts across computer applications and files of several functional areas. In the majority of the installations, the EDP manager is forced to flatly turn down the manager's request. He either cannot respond because the needed data is hopelessly locked up in existing poorly documented computer files, or because responding would throw the normal computer processing into upheaval.

In a minority, but growing number of other installations, however, the EDP manager is increasingly able to effectively respond to the manager's ad hoc request for information. These EDP managers have been implementing a data base approach through employment of data base software. At this point, extreme caution of interpretation and extrapolation is warranted. Data base or DB rings too much like the illusive Management Information System, or MIS, of the past and the present. It clearly does have some of the same characteristics, but as I will demonstrate in this paper, it is founded on a much more sound footing. I will also provide evidence that the forces are in irreversible motion. But similar to other major computer innovations, the early implementation is expensive and inhibited with organizational change problems. These problems should not be given short shrift, nor should the interpretation of their significance be allowed to overshadow the direction that has been set in motion by the technology.

AN HISTORICAL STAGE PERSPECTIVE

A recent research study revealed that the EDP budget for a number of companies, when plotted overtime from initial investment to mature operation, forms an S-shaped curve.[1] An analysis of the events associated with the S-shaped EDP budget curve led to the formation of a stage hypothesis of EDP growth. Exhibit I shows the S-shaped EDP budget curve pattern and the three growth processes that must be dealt with as an EDP function matures: (1) growth in EDP management techniques, (2) growth in specialization of personnel, and (3) growth in computer applications.[2]

Because of the rapid growth of computer technology, management of data has developed haphazardly and in a laggard fashion over the years.[3] A general approach of data management has emerged only very recently, and, consequently, applications have developed discretely from one another in an unintegrated and wasteful fashion. Further, each increase in the complexity and capabilities of computers has brought new generations of applications, but these applications still, for the most, part, have been specialized in nature, designed for a specific operational use or for a specialized staff function.

Hence management of data has continued to develop in fragmented fashion and at rather low organizational level—at a subdepartmental or substaff level.

Today, upper levels of management are seeking information that can be generated only from properly structured, company-wide pools that include data from the narrower applications located further down in the organizational hierarchy. That is, management information today requires that a company have a data base which can be used in conjunction with broad-range programs, to generate information on a broader and more comprehensive scale than the single, isolated applications of the past could usually do.

Notwithstanding the new demands, tradition is still strong; indeed, it has barely been challenged. Exhibit II represents the traditional way of doing things—collecting and coding data for specific programs and thereby gluing them more or less permanently and exclusively to those programs. In retrospect, this approach has had three significant disadvantages.

Files and records have tended to become redundant

Suppose Company X originally had only a single, computer-based system—say, for accounts receivable—which is represented in Exhibit II as Program I.

Program I has three data files: A, B, and C. File A contains customer records, each consisting of data elements a and b; a might be the customer name and b his outstanding balance. Files B and C contain other data elements needed for the accounts receivable program.

Assume that now the company wishes to implement a second program—Program II, as illustrated in Exhibit II—with Files D, E, and F, comprehending elements a, b, c, d, f, and g. Note that the company already has all these elements, except g, on file for Program I. In all probability, however,

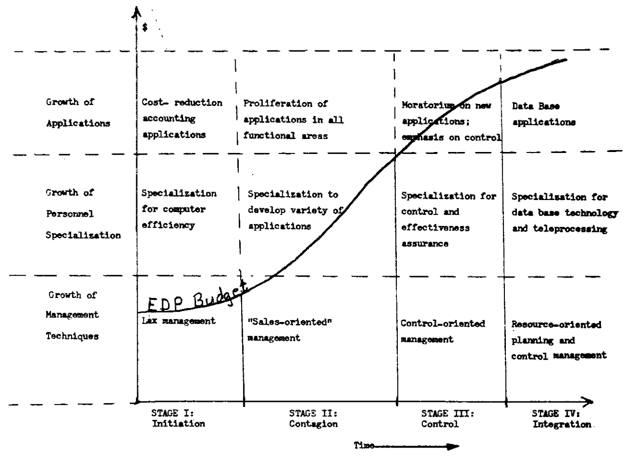


Exhibit I—The four stages of EDP growth

its programmers coded Files A and B (including all the elements a, b, c, and d) expressly for Program I, and hence cannot now use A and B intact for Program II. Thus the programmers have to make a choice:

(a) They can recode A and B so that these files can be used by either Program I or Program II. But this would mean rewriting Program I to take account of the recoding.
(b) Alternatively, they can build two "new" files, consisting of data from A and B but coded for the special convenience of Program II.

In the past, when faced with this kind of choice, an EDP department has usually just gone ahead and constructed the two "new" files. Going back over Program I ordinarily seems like too much trouble, so making up the new files seems the easiest way out. It is—in the short run.

But in the long run, as the exhibit shows, Company X might easily find itself creating more and more quasi-duplicate files as it adds new programs. For example, it will need two new versions of File B for Programs II and IV—that is, Files E and K. It will need three new versions of File A for Programs II, III, and IV—that is, Files D, G, and J. It will need a new version of File I for Program IV—that is, File L. And so on. The redundancy of data is obvious. In just this little, highly simplified example, 7 out of 12 (58 percent) of the data elements of the files are redundant.

Initially, redundancy does not cause a great deal of trouble. As soon as pieces of data must be updated, however, it does cause a great deal of trouble. In an EDP department of any size, it is virtually impossible to update all the redundant files and reports in systematic and synchronized fashion. Consider what must happen if Company X adds a customer: it must update A, B, D, G, and J, and that would only be the beginning.

Once files, records, and reports have begin to overlap and updating becomes a serious chore, updating procedures begin to sag of their own weight and different parts of the organization begin to receive inconsistent reports generated from files that are in various states of disrepair. In one large company, the inconsistencies between sales reports at the division level and sales reports at the branch level were so extreme that the salesmen began to keep very elaborate manual sales records. These two sets of reports were, in fact, generated in large part from redundant files that were updated at different times.

These particular inconsistencies resulted from a mere difference of organizational level—that is, the divisional versus the branch level. Severe redundancy problems can arise even more easily when reports from one function must be meshed with reports from another function. For example, there is absolutely no reason to expect that a company's inventory-control report will jibe with its accounting report unless the updating disciplines for the files of both functions are synchronized with each other.

Even slight variations in the data used for the two functional reports can cause glaring inconsistencies:

In a large retail chain whose applications had developed in the traditional fashion, the needs of the business forced management to request the integration of a number of different functional programs and systems. With great effort, the job was done. However, it was done in such a way that many quasiduplicate files were created and many separate, but essentially similar, programs were patched together. The company suddenly found itself spending 90 percent of its programming man-hours just keeping the programs running in concert and the files up-to-date.

At the very least, redundancy, spells confusion and expense for any sizable operation. Perhaps its worst feature is

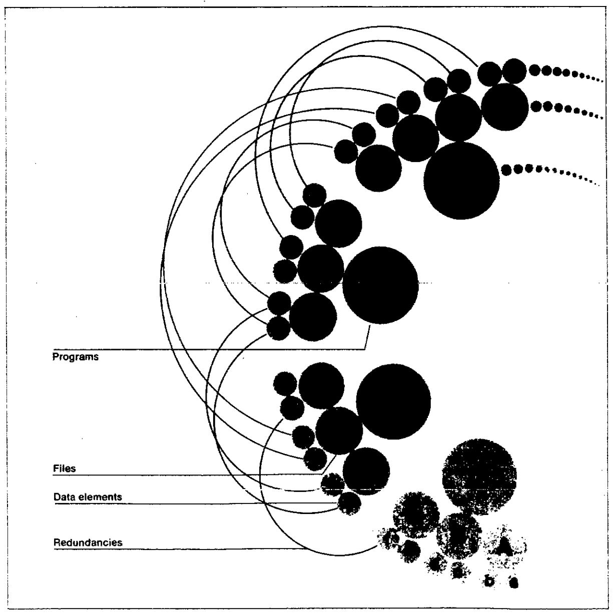


Exhibit II—The traditional approach to programs and data

that the longer a company follows the traditional pattern and keeps adding new programs and redundant files of data, coded specifically and exclusively for those programs, the greater the task it must face when it finally assembles all its data in a single pool, so structured and coded that new programs can be run without extensive recollection or recoding of data.

The traditional approach undercuts or aborts the advances of computer technology

Originally, the relatively high cost of on-line storage was a main factor that induced companies to delimit the scope of programming and therewith the amount of data needed during any given run. In effect, this reinforced the practice of creating and maintaining separate files for each application in the company's portfolio—companies tended to store no more data than were needed for the run at hand.

Today, however, many companies that have followed the traditional route, but have acquired up-to-date on-line storage systems, find they have the capacity to keep relatively huge amounts of data alive in the system. But their data are still organized and coded along first-generation computer lines—that is, by specific programs. From a rational viewpoint, this is as awkward, expensive, and absurd as keeping modern accounting records wholly in Roman numerals.

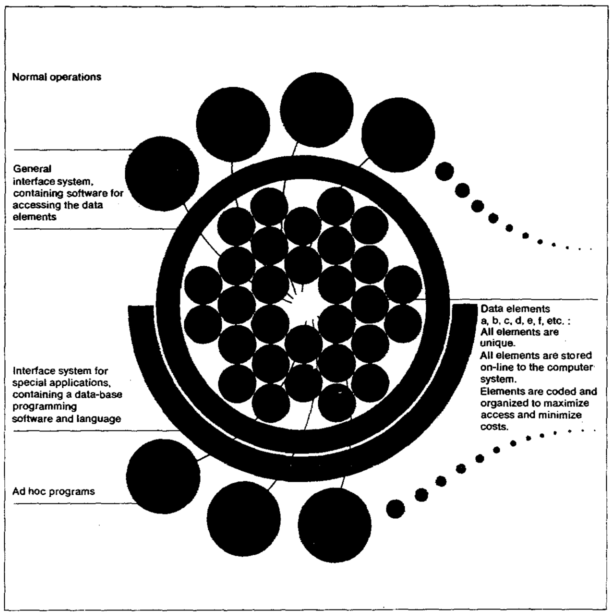

Normal operations
General interface system, containing software for accessing the data elements
Interface system for special applications, containing a data-base programming software and language
Ad hoc programs
Data elements a, b, c, d, e, f, etc.: All elements are unique. All elements are stored on-line to the computer system. Elements are coded and organized to maximize access and minimize costs.

Exhibit III—The data base approach to programs and data

The traditional approach obstructs upper management's growing demands for applications that require a data base

A review of the evolution of computer-based applications runs as follows. First, the computer was first used to replace existing manual functions, primarily within the accounting function. Next came the integration of computer-based systems within and between functional areas. Now cross-functional/interlevel systems are being developed to serve middle and upper-middle management; or, to put it another way, management is now demanding the benefits of computer innovations. The redundancies and inefficiencies that result from the traditional approach to the management of data become so signal and so extensive that applications can be adequate only if they are developed in such a manner that specific programs are separate from the data.

THE DATA BASE CONCEPT

As Exhibit III shows, the data base concept is to structure a company's computer-readable data into a single pool, which is used to run both routine programs and programs written in response to ad hoc requests. Note that no files appear in this exhibit. The base of data elements and their *structure* supplant the specific files. Note also that two additional software systems are in evidence here which were not in evidence in Exhibit II:

(a) The data base interface system enables a specialist data base programmer to organize and structure the data elements in a manner that minimizes or eliminates redundancy and optimizes the economic costs of data storage and accessibility.

(b) The interface system for special programming includes a high-level programming language especially designed for manipulating data elements contained in the data base, solving problems, and producing reports. To write ad hoc programs, the programmer works successively, through the interface for special applications and the general interface system to the data base itself.

Comparing Exhibits II and III, one can see an immense contrast between the traditional concept and the data base concept, both theoretical and practical.

Since much of the computer technology necessary to implement the data base concept exists and the rest of the technology is being developed rapidly, a strong case for adopting the data base approach can now be made. Yet, in operational terms, the concept is still novel. To what degree is it being used? What are the issues and problems involved in implementing it? By what strategies can a company work toward a data base? And what benefits can we realistically expect from it?

AN INTERVIEW STUDY

To answer these questions I administered a pattern interview to the data-processing managers of ten companies in six diverse industries. The questions permitted unrestricted responses, and hence the information these managers provided is not as clear-cut as one might wish. However, it is informative. The opinions expressed varied considerably among the EDP managers. By and large, a given manager's opinions reflected the particular stage his company had reached in the evolutionary progression toward full use of the data base concept.

First of all, I found a certain amount of confusion about what "data base" means. My open-ended question, "What is the data base in your company?" usually brought first a puzzled expression to the manager's face, and then a request for clarification. I answered that I wanted a statement on how he views his company's data base, if, indeed, he views it at all.

Responses ranged all over the lot. Some managers included all the computer-readable data in their company. Others defined the base more narrowly—for example, as including only the random-access disc files used for routine reporting and analysis.

The common thread in the responses was "computer-readable." Since all the interviewees were data-processing managers, this common thread is not surprising. But, obviously, the great majority of an organization's data are non-computer-readable; they are maintained in file cabinets as well as in the minds of management.

Although more and more data are being put into computer-readable form, as the technology improves and makes more sophisticated computer-based applications both feasible and economic, much of the literature on data bases falsely assumes that companies have already translated all the data needed for these applications into machine-readable terms. This simply has not yet happened—indeed, most companies have not even begun to collect the data needed for these applications, in machine-readable form or otherwise.

In general, the more advanced a company's use of the data base approach, the less naive and more realistic the manager's definition of what the base ought to contain—for example, "shared random-access files used for [periodic] production programs and ad hoc management requests." Such a definition reflects the two key characteristics of the data base: (a) sharing data between programs, and (b) structuring data so that ad hoc management requests can be served.

One data-processing manager articulated the criterion of responsiveness to ad hoc management requests especially well. He said that his company will realize the data base approach fully when he has incorporated the technology that will permit him to respond to any reasonable request by management for reporting or analysis within one day, and without undue degradation of his continuing data processing. He further described a reasonable request as one that draws on existing computer-readable data.

Responses to my questions on the structure and integration of computer-readable data grouped into three categories. On one end of the spectrum were the companies that structured their computer data on the basis of individual applications, with only limited cross-functional integration among applications—that is, sharing data between such functional applications as manufacturing and accounting. In the middle of the spectrum were the companies that had aggressively designed applications for cross-functional data sharing. On the other end of the spectrum were two companies that not only had achieved a high degree of cross-functional integration of their data base, but also had begun to achieve an interlevel integration—that is, an integration of their data base for managerial reporting and analysis with their operational data bases.

Both of these companies used commercial data base software packages. They used one software package for data organization (the General Interface System of Exhibit III), and a different software package to produce ad hoc management reports and analyses (the Interface System for Special Applications in Exhibit III).

Both data-processing managers were reasonably satisfied with the commercial software they were using. Nevertheless, they both commented that even the most sophisticated data base software commercially available did not incorporate the more advanced data-structure methods. Such methods coordinate theoretical data structures (for example, things resembling immense decision trees) with the access constraints of physical storage devices, such as rotating magnetic discs. Suffice it to say that data organization is extremely complex and technical.

It is so complex, in fact, that one is virtually forced into using commercial software. One of the data-processing managers stated that structure technology is so complex today that he could not possibly support an in-house effort to develop the software. The other manager had initially hoped to develop his own data base software, but, after a preliminary investigation of the costs and problems, he decided to acquire a commercially available package.

However, this complexity ultimately derives from the nature of the key tasks for which top management wants the data base to be used. If upper management focuses on key tasks that embrace all the company's data and require very extensive vertical and horizontal integration of reports and analyses, the job of organizing the data base is tougher than when the key tasks embrace only a part of the data and require less than the complete integration of all functions.

The EDP managers interviewed expressed concern over these major organizational issues associated with the data base:

(a) Acquiring personnel that can handle its technical aspects.
(b) Funding and developing suitable charge-out systems to support it.
(c) Setting and enforcing company-wide standards.
(d) Using the data resource to best advantage.

The major associated technical issues for which they expressed concern were these:

(a) Converting data to data base form.
(b) Providing appropriate software for the interfaces.

(c) Designing a data base which will permit ad hoc responsiveness without degrading normal computer processing.
(d) Building in reliability and the ability to reconstruct lost data.

Both the organizational issues and the technical issues were generally felt to be of such magnitude that aggressive action on implementing the data base concept fully was not warranted at the time. The consensus was that the concept is sound, but that much more needs to be done administratively before it can be effectively realized in practice.

MANAGEMENT ACTION[4]

Use of the data base concept is the next natural milestone in the evolution of EDP applications. It embraces the specialization of EDP functions; it allows management real flexibility in satisfying its need for information; and it permits companies to view and use their data as a real resource.[5] Yet caution and patience are advised in pursuing the concept. What should managers do to deal with this push-and-pull condition?

Take the idea seriously

Upper management should provide direction to the EDP manager by identifying key tasks of the business and setting priorities for an improved information capability. Perhaps the single most important factor to break out of a parochial treatment of data is upper management's guidance and its insistence on exploiting data for the interests of the business.

Set up a data administration function

The issue is when to set up a data administration function, rather than whether to have such a function. Ultimately, an administrator will be needed to implement the data base concept, anyway. For those companies currently without such a position, an administrative structure is needed for formulating a data base implementation plan, and establishing data base standards, controls, and access procedures. At a minimum, a data base specialist should be acquired now to provide decision-making guidance for the EDP manager and steering committee. This person can also provide guidance in evaluating and selecting appropriate software.

Incorporate data base technology into the computer system

The hardware technology, as well as the software technology, for data bases has matured to the point that the data base concept can be both feasible and cost-effective for many organizations. While the company will not be noticeably hurt in the short run by ignoring data base technology, it will in the longer run.

Also, the data base concept cannot be implemented overnight. If a company begins to plan and act now, it can assimilate even drastic technological improvements into its existing systems in an orderly fashion.

To incorporate the technology that will permit data base operations, an organization must identify its key computer-based systems and restructure them (a) to remove redundancy, and (b) to facilitate their use by higher levels of management. For the present, companies must probably acquire commercial software for structuring data and responding to management requests for ad hoc analyses and reports.

Think of data as a resource

For the longer term, management should begin to think of data as a basic resource. It should accept this idea as a natural consequence of functional specialization of the general management function. Since the data-resource concept is closely associated with a fast-moving computer technology, management should expect to see the movement toward specialized data-management activities proceed at a faster rate than, say, specializations in the human resource function.

In retrospect, the curve shown in Exhibit I takes the same shape as the generic learning curve. In large part, I think that the curve has been driven by developments in hardware technology in the second and third generation computer systems. Now, however, the advancements seem to be taking place more in software than in hardware. The breakthrough most likely to start off another S-shaped EDP budget curve is data base technology. In dealing with data base technology, caution is in order and the painful lessons of the past should not be cast aside with the first blushes of enthusiasm in perceiving the potential of the concept.

REFERENCES

1. Nolan, Richard L., "Managing the Computer Resource: A Stage Hypothesis," *Communications of the ACM*, Vol. 16, No. 7, July 1973, pp. 399-405.
2. Gibson, C. F. and Richard L. Nolan, "Managing the Four Stages of EDP Growth," *Harvard Business Review*, Vol. 52, No. 1, Jan.-Feb. 1974, pp. 76-88. Gives a detailed discussion of the stages and the organizational issues for each stage.
3. Nolan, R. L., "Computer Data Bases: The Future is Now," *Harvard Business Review*, Vol. 51, No. 5, Sept.-Oct. 1973, pp. 98-114. The ensuing discussion about the way in which computer data has been treated in the past, and the explanation of the data base concept is largely taken from this article.
4. Ibid., pp. 113-114.
5. Nolan, R. L., ed., *Managing the Data Resource Function*, West Publishing Company, St. Paul, Minnesota, 1974. For an extended treatment of the evolution of the data resource function.

Data bases—Uncontrollable or uncontrolled?

by CHERYL M. TRAVER

Stanford University
Stanford, California

In the last two years the "data base" concept has received as much management publicity as did "Management by Objectives" in the years just prior. The popularity of and recognition given to the power of computing in recent years, coupled with increased awareness of the role the enlightened manager should assume, have encouraged one installation after another to acquire a data base system in the pursuit of modern techniques for information retrieval. As both hardware and software have become cheaper and more widely available, the decision has become easier to make; thus the apparent need for foresight and planning has declined. Unfortunately, all too many people are now seeing the entire data base technology as an impossible dream and discrediting the concept as an expensive toy devised by the ever-hungry computer vendors. This trend is summarized in a brochure for a current seminar on data bases: "No subject in computing history has had the import of data base, nor disaster stories of equal magnitude."* And yet the "disasters" are not necessary. With proper foresight and planning and continued supervision, a data base system can produce all the results it promises.

As long as the concept was new, the software expensive, and the expertise nonexistent, much planning was done before anyone ventured into the new territory of the data base systems. But as the successes were publicized and the software became more available, many installations began to acquire such systems with little more thought than new compilers and card readers. Is it any wonder then that fiascos occurred and disillusionment set in? It is unfair, however, to attribute these disasters to the data base technology. One must look instead to the people behind those systems, and in particular to management. For, in the new era of data base technology, management cannot blame the system, or the computer, or the people pushing the computer buttons or coding the computer programs. Data base systems die (or become diseased) through lack of management interest, lack of controls and standards, lack of committed resources, and the absence of a cohesive bond which encourages the users of data to define their needs and communicate their desires.

As stated in the abstract, data bases are a lot like children. Both carry a great deal of appeal to the casual observer, probably due to the fact that they are usually presented at their best by a devoted set of admirers who extol their virtues and minimize their shortcomings. Vendors out for a sale will, of course, never stress (perhaps not even mention) the limitations of a system. The owners of such systems, on the other hand, are similar to the man who, having nurtured back to health a scraggly mutt that was left on his doorstep, becomes convinced the bastard is at least equal in worth to the pedigreed schnauzer next door. Disaster stories and problem children are presented in third-party terms and are interpreted as happening only to the "other guy".

Furthermore, the owners of many data base systems are a lot like many parents. They are unhappy, disillusioned, and cannot understand how they acquired an uncontrollable and unmanageable offspring. They complain of the exorbitant investments of resources, time, and money, and wonder why they ended up with a rotten apple. They blame the environment, God, and country—everything but themselves.

Fortunately, though, the concept of parenthood has been around long enough for us to realize that model children are not accidents, but rather the product of hard work and concern. In recent years many books, seminars, and classes have appeared which aim to help parents and potential parents acquire the necessary knowledge. On the other hand, typical of a field that is growing very rapidly, the technology for supporting the sophisticated data base was available long before the guidelines on how to use it. Indeed, the early users were often not even aware of the potential problems. Many of such systems were dismal failures or at least disappointments. There were some, of course, that survived and even prospered, usually as the result of a gifted and farsighted individual. The ratio of successes to failures, however, strongly indicates that data bases are a risky business. The publicity of the failures has frightened many and caused others to blame the computer and its software. For the farsighted, however, the successes prove the benefits can be had—if one can learn how to obtain them.

THE DECISION

First of all, it should be recognized that data bases are not for everyone. Compared with the traditional fixed

* *Data Base 1974 Seminars: Fact vs. Fiction,* Performance Development Corporation, 1974 brochure.

input/output tape systems, data base systems are expensive. The very sophistication and complexity which permit the flexibility and versatility of the new type of system also make it difficult to work with. When there are many options, mistakes are more frequent, since there are also more wrong solutions. Mistakes mean money since they require rework—manpower and machine time.

Data base systems usually include online terminal facilities for retrieval and/or maintenance of data. These facilities encourage increased cognizance at the user level. Since the data are directly available, the operating environment of the computer installation is forced to become more polished. Mistakes can no longer be covered by reruns without user knowledge. The support for an online system requires actual dedicated machine time and thereby substantially reduces the wall-clock hours available for batch work. The batch capacity of the CPU is therefore greatly reduced.

Ongoing support

The new sophisticated systems also require ongoing support. They are not like the "canned" payroll packages or budget accounting systems which, although usually requiring an initial conversion of input/output documents, may be executed week after week with minimal intervention. The good data base system is alive. It needs supervision and care in order to be molded and tuned to react better to the needs and desires of its user population.

One could compare the data base system to the rose. Although universally recognized as one of the most beautiful flowers, the rose does not appear in every garden, even though its cost is not prohibitive. The fact that, unlike the tulip or chrysanthemum, the rose bush requires constant care and attention makes many people give up that beauty and settle for other pretty flowers which require less effort.

Data base systems are open-ended—if handled properly they can be expanded to include an ever-widening spectrum of information needs. However, this expandability can be expensive. The tendency is strong to permit such systems to grow too rapidly. User desire and user need may become confused and often the installation which can barely afford to support the user needs finds itself in a costly situation where the desires of some users are being catered to before the needs of others have even been addressed.

These new systems can even be expensive in terms of computer hardware in that the search for more rapid means of data access for ever-growing online demands encourages a proliferation of different hardware peripherals.

Management commitment

However, the greatest expense of a successful data base system is management time. To be successful, a data base system requires standardization and compromises at the user level. In many environments, especially university administrative areas, time has permitted various departments to become semiautonomous. This characteristic is usually very evident in the automated systems of the installation where each application is an independent system with its own editing rules, formats, and procedures. In a data base system data must conform across areas in order to be able to form a base for information retrieval.

This need for conformity or standardization is similar to the process of American urbanization.* In colonial days the pioneers were self-sufficient and independent; they had no need for standardization. The shift toward urbanization and community living forced a certain amount of standardization: people had to learn to cooperate in order to live together in harmony. Thus was born a government of checks and balances. Likewise, the desire to make common use of data necessitates an acceptance of a form of "government" and this government needs definition of, commitment to, and enforcement of "laws". Although the rules can be (and all too often are) formulated and enforced strictly by the computing organization, our society seems adapted to a more democratic form of rule. Hence, the management of all areas contemplating use of data base information must become involved.

Since the needs and desires emanate from the users, those are the people who should be determining the nature of their data and of the system to transform those data into information. This commitment of management time is heaviest in the early stages and will be difficult because no results will be immediately apparent. Many of the discussions may even seem absurd. For example, agreement on a standard format for a computer-stored name may seem trivial. However, without that standardization the resultant computer systems may not be able to search and match satisfactorily. Worse yet, the computer system may be burdened with the weight of trying to resolve through software those problems which should have disappeared through user discussion.

Unfortunately, most installations are not first entering computerization when they contemplate a data base environment. They already have precedents, conflicting standards, and users who have become accustomed to their own way. It is the responsibility of management to see that these users come to agreement. Otherwise, they should not show surprise if the computer administration delays delivery of the data base system or if that system seems superficial or incomplete.

PREPARATION

Even when the commitment of time, money, and manpower has been made, not all data base systems are successful. The biggest reason is lack of planning. Having made a large commitment, the user communities understandably want results. The computer personnel, in an attempt to satisfy their users, too often rush headlong into the acquisition of a data base vehicle and the implementation of the

* John K. Lyon, "The Role of the Data Base Administrator," *DATA BASE*, Winter 1971, pp. 11-12.

first base systems. Somehow the estimates are always low and corners are cut in the attempt to placate the expectant communities. The first systems may even seem fairly satisfactory. This approach, although reasonably successful with the independent systems of the past, portends disaster for a data base system. The data base system is a self-propagating animal which feeds on the insatiable desires of its users. As more is learned about the capabilities of the system and the characteristics of the data contained therein, more is demanded. Widespread and diverse areas in the organization will become involved; new sources of information will be discovered. These conditions are healthy. However, if there were not strong foundations upon which the original data base was built, the additional demands may overtax the system. To avoid a short life for the data base, the long range goals, needs, and desires must be carefully analyzed even before the selection of the data base vehicle—and obviously before the design of the first application systems.

The data

The first element to be analyzed is the data themselves. The various types of data, both those which currently exist and those which would be desirable to obtain, should be analyzed. In-depth descriptions need not be detailed in the early stages of planning. However, those data which span traditional file boundaries must be isolated. (An example of this might be university instructor course load, which is used for personnel reports and also for student study lists.) These interrelated data may not seem critical in the early phases of a data base system. However, due to the tendency toward the development of more complex analyses and correlations which were unavailable in the past, these interrelated data must be structured within the data base in such a manner that future demands do not overburden the retrieval structures and/or security schemes of the new system.

All data should be classified according to their "owners" and "users". The owners are those people who can modify the contents or nature of the data—the traditional controllers of the file. The users are those people who are permitted to peruse the data. The owner/user relationships must be carefully studied to ensure that an appropriate security scheme can be evolved.

Usage patterns

Once the data have been analyzed and isolated, the retrieval and maintenance patterns of the data should be identified. These patterns will be used to determine the spectrum of data base features that are necessary and desirable and will help to determine the need for online retrieval. The 'currency' requirement of the data is also important: how current the data in the base must be to a large extent indicates the real need for an online maintenance environment. If the data can be a day or a week old, a pressing requirement does not exist. (Of course, online maintenance, given that online retrieval has already been justified by need, may be more cost effective and is usually more desirable than overnight batch maintenance.) Since terminal equipment is still rather expensive, it must usually be carefully justified by need (or by enough available funds to justify desire). In addition, the establishment of an online environment is usually a one-way street: very few organizations revert to batch-only systems.

The data base vehicle

The characteristics of the retrieval and modification patterns should be summarized and condensed into a spectrum of required features (those functions/characteristics which must be facilitated) and a spectrum of desired features (those functions/characteristics which would improve usage and expansion of the system). These spectra may then be compared with those features provided by the available data base vehicles.

A thorough analysis of the available budget and manpower for programming and support must be done, both for the current time period and for several years hence. By comparing available resources and desired characteristics with costs and offered features, the organization should be able to find a suitable data base system (or to decide that any such system is beyond the current scope of feasibility).

DATA ADMINISTRATION

After the organizational needs and desires have been defined and a data base vehicle acquired, a live data base should not be established until proper controls over that system have been established. It should be clear from the records of those data base vehicles currently available that each of them has the potential for a successful data base system—and the possibility of failure. The difference is not inherent in the data base vehicle itself but rather in the manner in which it is used—or misused. The data base system must be controlled, or it becomes uncontrollable.

The specific method employed to produce the controlled environment is not important: the presence of such a method is. However, the trend in recent years has been to focus the responsibility for this supervision in one individual within the computing organization. He may be called a data base administrator (DBA), data manager, information specialist, systems controller, information scientist, communications administrator, or any of a number of other titles. His functions may vary from the purely technical to the strictly political. His responsibilities are often almost infinite, his authority all too often nonexistent or only token.

If the responsibilities and functions attributed to the DBA are even cursorily examined, it should be obvious that no single individual, not even a "superman", could adequately handle all of the work. It should also be apparent that the specialties required vary from those associated with computer technicians to those needed to converse with and instruct clerical personnel to yet others directed at the involvement with middle and upper management. This diversity and breadth of functional responsibility is finally being recognized. A few installations have already begun forming a Department of Data Administration so that the diverse needs may be satisfied via more than one individual: perhaps a teacher, writer, technician, analyst, and user representative. A strong manager is mandatory to ensure consistency and the strong commitment that the user is all-important.

However, regardless of what the organizational structure is, a consistent attitude is shared by these installations: the recognition of the need for control over and monitoring of the data base. A summary of the responsibilities and functions follows.

Maintenance of the data base and its directories

One of the most recognized characteristics of the data base environment is the fact that control must be exercised over the creation, deletion, and reorganization of files. Most data base systems have sophisticated file structures which combine traditional files into a single data set, often on the same physical device. Many support chaining between logical files and even permit references to multiple records in a manner that is transparent to the requester. Caution must be employed when deleting or reorganizing one logical file since other systems may depend on its contents. The allocation of space for new files must be coordinated to ensure that overlapping does not occur and that usage on the various devices in the data bases can be balanced across all user areas.

The system files used by the data system must also be controlled. This usually involves the maintenance of a system dictionary or directory file and a file which contains data on the space utilization and boundaries of user data files in the data base.

As will become increasingly more obvious, the very sophistication which makes a data base system flexible and powerful also permits accidental catastrophes and seemingly self-destructive files. Many an installation has been forced to recognize the need for centralized control on major modifications to the data base when two programmers build a file in the same allocated space or when the system directory is destroyed by an inexperienced operator.

Control over the contents of the data base directory is also important, since that is usually the user means of access to his data. Because the definitions of all data items within the entire data base are stored within one directory and users may often gain access to more than one file, the names and structures within the dictionary must be monitored to ensure compatibility, uniqueness, and consistency with the names in the overall system. Revisions to the directory must be controlled for the same reasons demanded by the physical data themselves.

It is also necessary to organize the printing and distribution of the contents of the dictionary. Frequently, Data Element Dictionary manuals, based on the stored contents of the physical dictionary, are prepared and distributed to the authorized users; it is, of course, very important to ensure that each user gain access only to those Dictionaries which contain data he may retrieve.

Security

Probably the duty most widely associated with the data base administrator is that of controlling the security aspects of the data system. Even in those environments where all the information in the data base is published in public documents the security requirements are stringent. This characteristic is no doubt precipitated by the new retrieval capabilities commonly offered the user: the more ease he enjoys in being able to analyze his data, the more apprehension he has about the ease others might share in accessing *his* data. As will be stated again, those requirements placed on a data base system because of user apprehension should receive the utmost consideration in that the primary purpose of most such systems is to provide users with more and better information.

Included under the enforcement of security are the assignment, reassignment, and modification of passwords and/or passkeys. Although the DBA normally supervises the password procedures, the policies and procedures governing the issuance of these passwords should be the responsibility of higher management or a consortium of users. Procedures for detecting the compromise of security keys and control procedures within the actual machine room may also be assigned to the DBA.

Integrity

The integrity of his data is very important to the owner of a file. The stability of the system with which he interacts is critical to the user of a data base. These needs are fairly easily met in the traditional computerized systems—and, even if not quite met, infringements can usually be minimized before reaching the user level.

In an online system, especially one which permits online updating, stability is a very elusive quality. Procedures on backup and recovery, logging systems and audit trails must be carefully established and even more carefully controlled. Backup hardware may even be needed, since the user who trusted his data processing department enough to discard his manual files will want to revert to them when he cannot get at his data because of a malfunction. Because of the integrated nature of the new files, testout procedures must be established and monitored for both software system modifications and new application systems.

The standardization of the codes among files is also related to data integrity. Inconsistencies between traditional files were easily tolerated in that only the computer program need know. But when the user begins to input his own retrieval requests he becomes easily discouraged if confronted by such situations as name being stored surname first in an admissions file yet surname last in the student master file. Unfortunately, although it is easy to convince users that consistency in codes and data content is critical, it is almost impossible to get them to agree on a change to *their* standard.

Quality control and performance measurement

In addition to the need for standardization of data content, the data base systems require a certain amount of standardization of application design, especially where terminal interaction with a user is involved. To the user all communications derive from a common source—the computer—and he will not respond happily to major differences in the responses requested of him.

In addition, since data base systems frequently support multiprogramming and concurrent accessing of data, lack of efficiency in a given program may affect the performance of another application or even of the entire system. Thus, standards must be established for application systems and these standards must be enforced by someone outside of the environment which always demands too much in too short a time.

Because of the user cognizance and the tendency toward sophisticated system activity, performance measurements become an important responsibility of the data base administrator. Hardware and software monitors may be used to measure CPU and hardware utilization and the distribution of access. For those systems supporting interactive devices, system and user response times should also be measured. The activity by terminal user is informational and often required for accounting interfaces. Such measurements may be used to assist terminal users who show abnormal usage. The performance of the system expressed in quantitative form often dispels user convictions that the system is slow or that "everybody else" is benefiting more.

Consulting and training

Because of his exposure to so many aspects of data management, the DBA often serves as the consultant, instructor, and technical advisor for his installation. Programmers and analysts need assistance in design problems and may also need help in program debugging under the software system. In addition to knowing all things about the data base system in use, the DBA must keep current on the "state of the art" and knowledgeable of other software and hardware, especially the data base packages on the market.

Programmers and analysts in the installation require much training in the techniques for data definition, file design, security features, application design, coding techniques, and—of course—installation standards. All of these guidelines must be incorporated into formal documentation.

Users and management must also be educated in the use of the data base and its features. Even using the terminal imposes a threat to many, in that they feel the pressure of the "meter running" and some even fear that "something" is looking out from inside the terminal screen.

User liaison

Communication with the user is the single most important role of the DBA, since in the long run it will usually be the reactions and attitudes of the data base users that will 'make or break' usage of a data base system. If the user is displeased or confused, he will not be receptive to a more sophisticated system. If he does not trust the security features or feels frustrated by his terminal, he will yearn for the old, more comfortable procedures. For these reasons the user attitudes must be considered, no matter how trivial or ridiculous. Frequent interactions and status reports, immediate assistance with terminal problems, and a log of trouble calls and user problems will help to comfort the unfamiliar user. Since interactions with the nontechnical populace are frequently difficult for computer personnel, the DBA may be the sole source of such 'hand-holding' talent.

IMPLEMENTATION

Once the procedures for sustaining a good data base have been established, the organization can begin to implement its own data base systems. Here, however, caution must be employed. Data base systems are not easy to design. Online systems are a new technology and not much expertise is yet available. The new systems are used so directly by non-computer-oriented people that the primary emphasis must be placed on an audience unfamiliar to the computer professional.

The greatest problem usually becomes apparent in the use of the data base vehicles themselves. The new software is complex and capable of sophisticated applications. However, it is nothing more than a tool: it must be used with skill and expertise in order to produce satisfactory results. A data base vehicle is like a typewriter. Although it is a well-established fact that the typewriter has greatly increased the productivity of secretaries, no one would expect anything but a disaster if an untrained person were required to exchange longhand for a typewriter. Yet most installations expect far greater miracles of their computing organizations when a data base vehicle is introduced. And, unfortunately, since most computing organizations do not realize the complexity of the new technology, the users of the first system may become the victims. Furthermore, although there are many fine secretarial schools which can consistently produce speed typists in a short amount of time, there are very few ways to learn data base technology and no ready substitute for one to two years of hands-on experience in a production environment.

As a result, the disaster-prone patterns are becoming familiar: The computer processing department plunges directly into the design and implementation of a major application—one with a high exposure profile and importance to the entire community. In an attempt to show results quickly, analysis is cut short, user interaction is curtailed, and the less visible features (such as edits and controls) are delayed "until later". There is little time to spend in training the programmers in the use of the new data base vehicle. The resultant system is obviously weak, but the lack of efficiency and shallowness of design are not always apparent until the second or third system is added. Eventually the hastily built foundations of the data base begin to crumble and work must begin again.

Thus, it is wise to approach the implementation with caution. Work should begin with a small low-profile application and expectations should be kept low. Above all, it should be recognized that not all of the problems are due to computer professionals. A true story illustrates the point: Having recently joined the computer professional staff, one of our analysts was especially interested in making a good impression with the user community. When one of them called to complain that she had not received her Numbers Report that week, he assured her that he would check into the situation. When none of the programmers could figure out which particular report might be the missing one, he went to the user, who promptly showed him a neat file wherein he was astounded to find page after page of *memory dumps*. It seems that one of the report programs had always encountered problems and the programmer, in order to facilitate debugging, had appended a partial dump of memory to the printouts of the reports. The corrections had finally been made and the dump option removed. The unquestioning user had simply filed the copies and worried only when the Numbers Report failed to appear.

Interconnection networks—A survey and assessment

by KENNETH J. THURBER

Sperry Univac
St. Paul, Minnesota

INTRODUCTION TO INTERCONNECTION NETWORKS

As the level of complexity of digital systems increases, the problem of interconnecting subunits is receiving increasing attention. We are reaching the point where processing speed cannot be further improved through the use of faster componentry. Further speed-up of systems will most likely result from changes in the organization and structure of hardware, rather than by raw circuit improvements. Another factor increasing the complexity of systems is the arrival of cheap, powerful LSI microcomputers which allow system construction involving a plurality of processors connected together to perform a specific task.[1,2] Restructurable system concepts are also very promising, but require extensive amounts of interconnective capability.[3] Thus, bus structures[4] are attracting considerable attention. This paper focuses on a small segment of the general bus structure problem; namely, interconnection (permutation, sorting, etc.) networks.

Figure 1 indicates the essential elements of an interconnection network. The network consists of a set of n input lines, a set of n output lines, a block of connective logic, and a set of control lines. The control lines structure the connective logic such that the n input lines and n output lines are connected together in some fashion. Clearly, if we allow all permutations of the line connections to occur there are $n!$ connections possible. Thus, an upper bound on the network control lines required is $\log_2 (n!)$. There are a number of variations of such networks that have been recently proposed. This paper will describe each of the major concepts and will give a global comparison of the concepts as to their capability, throughput delay, allowable size, etc.

Typical applications of the networks will also be indicated. One interesting point to note is that since the functionality of the networks is so difficult to realize, little work has been done on the networks to make them practical other than trying to discover ways to cost effectively achieve their functionality.

Sorting via software techniques has been studied extensively. Martin[5] presents a comprehensive survey of the software sorting problem and its various solutions.

DESCRIPTION OF INTERCONNECTION NETWORK CONCEPTS

The most important interconnection network concepts will be described in this section in chronological order.

Benes' telephone network

Benes[6,7,8,9] may be considered the father of interconnection networks. He performed the first major in depth studies of the rearrangeable array and the two input-two output permuter cell. Additionally, he studied the networks in terms of their combinatorial and topological properties.

Benes defined a connecting network as an arrangement of switches and disjointed transmission links which allow a set of terminals to be connected together in various combinations. Benes also studied the rearrangeability of a network and the concepts of "blocking" and "distance." Rearrangeability deals with the ability to route new calls during the presence of current calls. A network is considered blocked, if a given pair of idle terminals cannot be connected. A network can be non-blocking in two senses. These are: (1) if rearrangement of present calls unblocks the requested call (non-blocking in the wide sense) or (2) if a network has no blocking states (non-blocking in the strict sense). Distance is the number of calls one would have to add or remove to change network states.

Benes uses graphs to describe his networks and derives necessary and sufficient conditions for them to be non-blocking in both senses.

A diagram of a typical rearrangeable network is shown in Figure 2. A Benes network for $n=8$ is given in Figure 3. Further extensive work on rearrangeable networks has been done by Joel[10] and Opferman and Tsao-Wu.[11] Opferman and Tsao-Wu indicate that extensions to $n \times m$ networks can be readily made using the rearrangeable array.[11]

Batcher networks

Batcher[12,13] networks were envisioned for use in sorting and merging applications and as a replacement for crossbar switching networks. A crossbar switch for n elements grows at the rate of n^2. A Batcher network grows at the rate of $(\frac{1}{4})n(\log_2 n)^2$ for n elements. The delay time for $n=2^p$ words is $(\frac{1}{2})p(p+1)$ basic element delays; i.e., $(\frac{1}{2}) \log_2 n((\log_2 n)+1)$.

Figure 4 shows the basic Batcher network element. It accepts two input numbers A and B and outputs their minimum (maximum) on the L (H) output line. Figure 5 is the symbol for a s by t odd-even merging network. The use of the comparison network to merge two ascending sorted lists $(a_1, \ldots,$

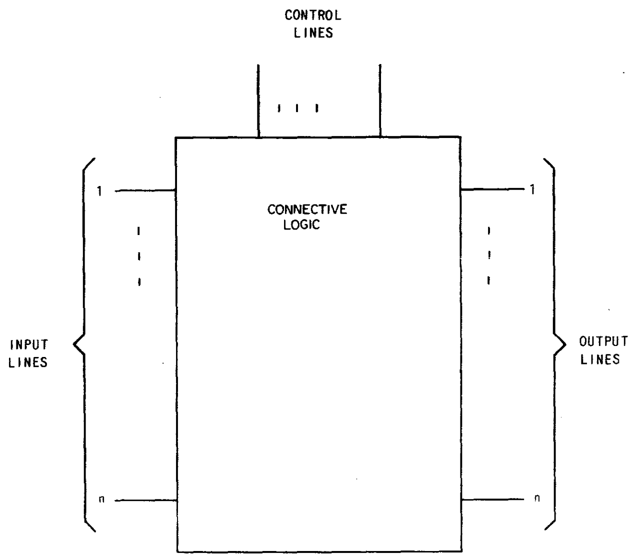


Figure 1—An interconnection network

a_s and $b_1, \ldots, b_t$) into a single ascending sorted list ($c_1, \ldots, c_{s+t}$) is shown in Figure 6. Since a 1 by 1 network is a simple comparator, Figure 6 is an iterative rule for the odd-even merging network's construction.

Another network (Bitonic Sorter) was also described. A sequence is defined as bitonic if it is the juxtaposition of two monotonic sequences, one ascending and one descending. Also it can be assumed that a sequence remains bitonic if it is split and the two parts are interchanged. Since any two monotonic sequences can be used to construct a bitonic sequence, a network that rearranges a bitonic sequence into monotonic order can be built and used as a merging network. Figure 7 gives the iterative rule for constructing a bitonic sorting network for $2n$ numbers using n comparison cells and two sorters for n numbers.

Sorting networks for arbitrary sequences can be constructed from either odd-even or bitonic sorters by forming ordered lists of length 2 and merging those lists two at a time, to form lists of length 4, etc..

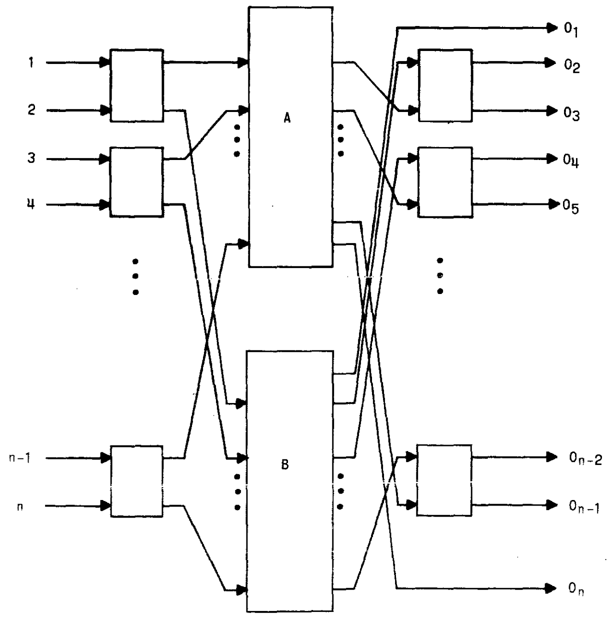


Figure 2—Benes' rearrangeable network

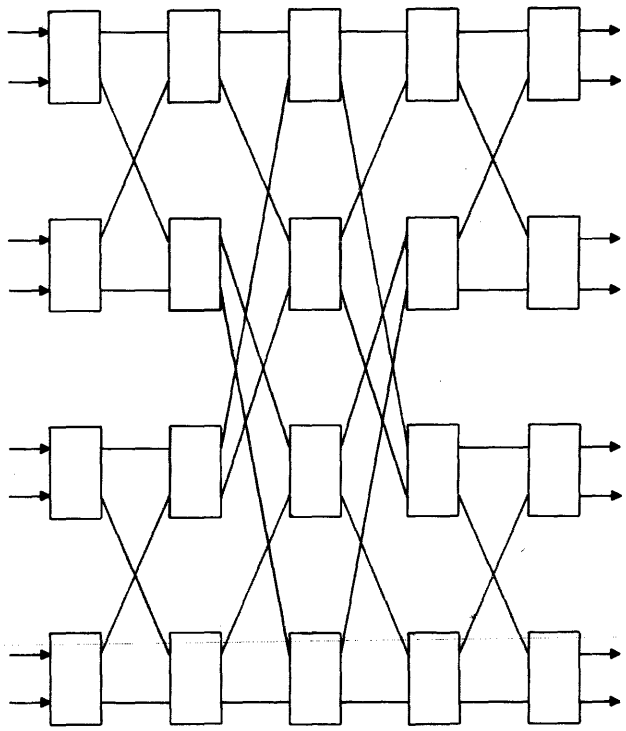


Figure 3—An 8×8 rearrangeable network

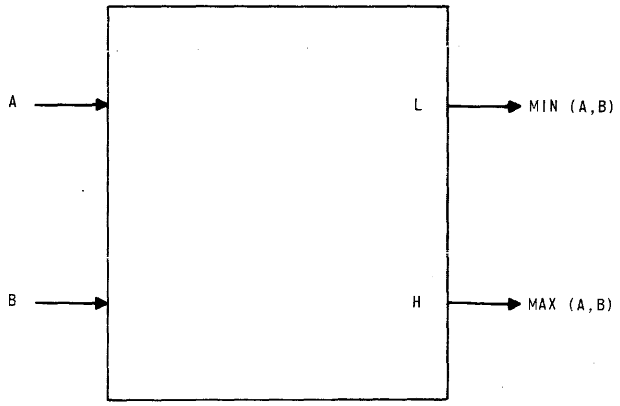


Figure 4—Basic batcher cell

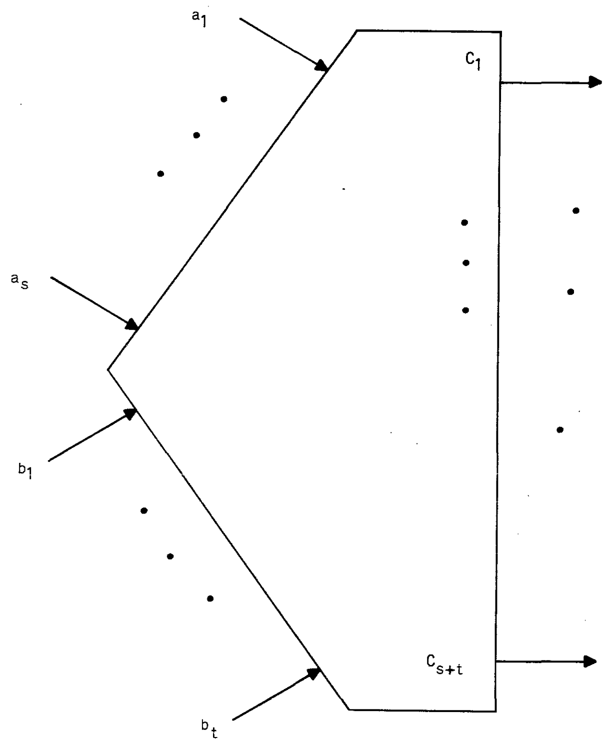


Figure 5—Batcher odd-even merger cell

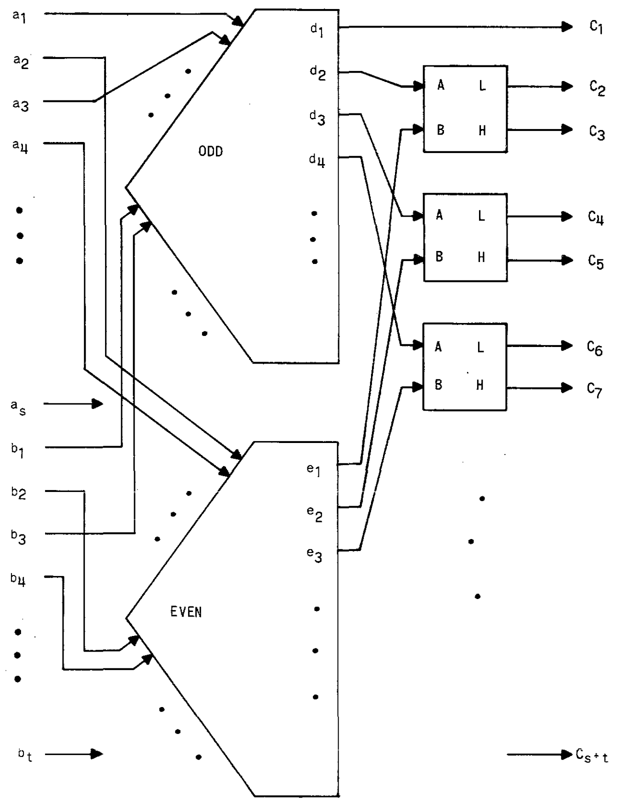


Figure 6—Batcher odd-even merging network

Kautz's cellular interconnection arrays

Much work[14,15,27,35,38,39,42,43,44,46,47] was done on interconnection networks at Stanford Research Institute. Kautz[46] provides a good summary of the results of the various studies. He describes three main problems. These are: (1) universality of interconnection networks,[38] (2) minimization of interconnection networks[39,47] and (3) permutation networks.[14] Problems (1) and (2) deal with the desire to be able to arrange a homogeneous set of elements in a "less-than-completely-connected" network to optimize some parameter. The universality of interconnection networks is studied by using networks represented as linear graphs. Problem (1) becomes the determination of the characteristics of universal networks which allow any configuration with a sparse interconnection structure to be realized as part of a "universal" network. For example, universal networks of six elements with interconnection links to no more than three other elements can be exhibited which realize every possible six-element network in which elements are connected to no more than two other elements. Problem (1) is abstracted to determining what graphs may be embedded in a universal graph with n nodes (elements) of degree (number of connections) d. Bounding relationships on n and d are derived. Problem (2) can be abstracted to trying to minimize the largest value of the number of PE's which one must pass through in communicating between two elements of a network. This problem can be considered in terms of the diameter of a graph. The problem

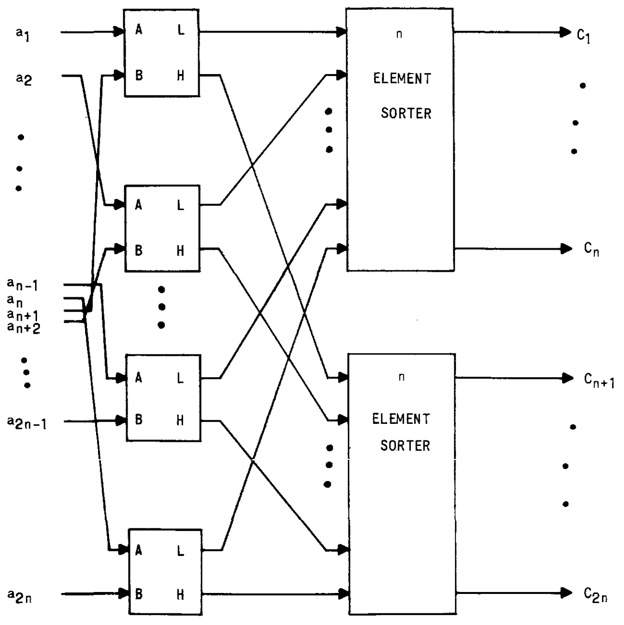


Figure 7—Bitonic sorter

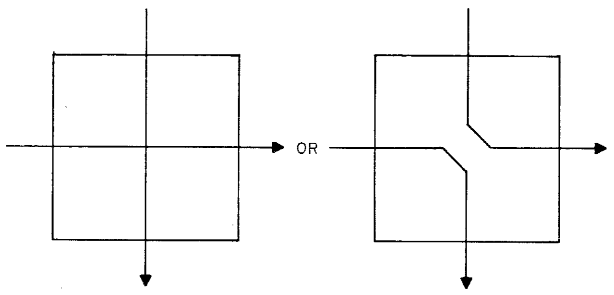


Figure 8—Basic Kautz cell

then becomes the synthesis of graphs with prescribed diameters but a constrained degree of a node. Various synthesis (construction) techniques and bounds are described.

Problem (3) concerns permutation networks. Kautz[14] et al. and Waksman[15] utilized the basic permutation cell shown in Figure 8 to build interconnection arrays. The cell has two states defined as "cross" and "bend." A number of different array types have been investigated, including triangular, diamond, rectangular, pruned rectangular, rhomboidal, square, almost square, Bose-Nelson,[16] and rearrangeable arrays.[9,14,15]

The triangular array is shown in Figure 9. The proof of its permutation capability is inductive. Clearly, the left most cell can perform the permutation for $n=2$. Assume that $n-2$ columns can perform any permutation of $n-1$ elements, then by adding column $n-1$ to the array the variable X_n may be switched into the sequence $Y_1, \ldots, Y_{n-1}$ at any point, thereby adding an element and making the sequence $X_1, \ldots, X_n$ map 1 to 1 and onto the sequence $Y_1, \ldots, Y_n$ in any fashion (permutation). The number of cells required is $\frac{1}{2}(n^2-n)$. The other arrays studied were variations that altered the level of interconnectivity and the number of cells in the arrays. Further, the regularity of the structure was manipulated and studied. The rearrangeable array was essentially an implementation of Benes' rearrangeable networks described earlier.[6,7,8,9]

Thurber's programmable indexing networks

Thurber[17] introduced a network called an "indexing network" to overcome some deficiencies of n to n interconnection networks. Limitations of interconnection networks typically are that input words cannot be repeated or deleted at the output. Blanks cannot be inserted into the output and the number of input words and the number of output words must be equal. The indexing network differs from the permuter in that input words can be repeated or deleted and blanks can be inserted in the output. For an indexing net-

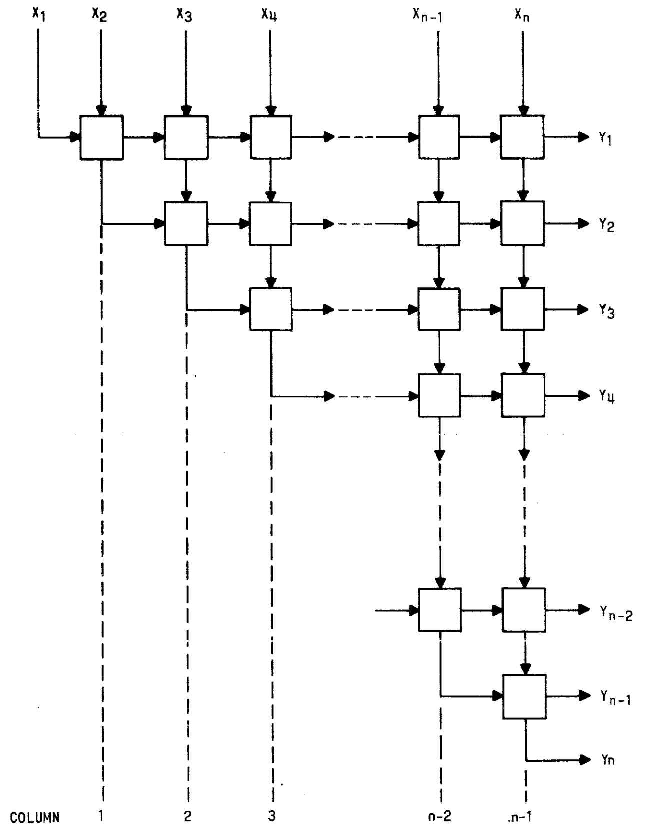


Figure 9—Triangular interconnection array

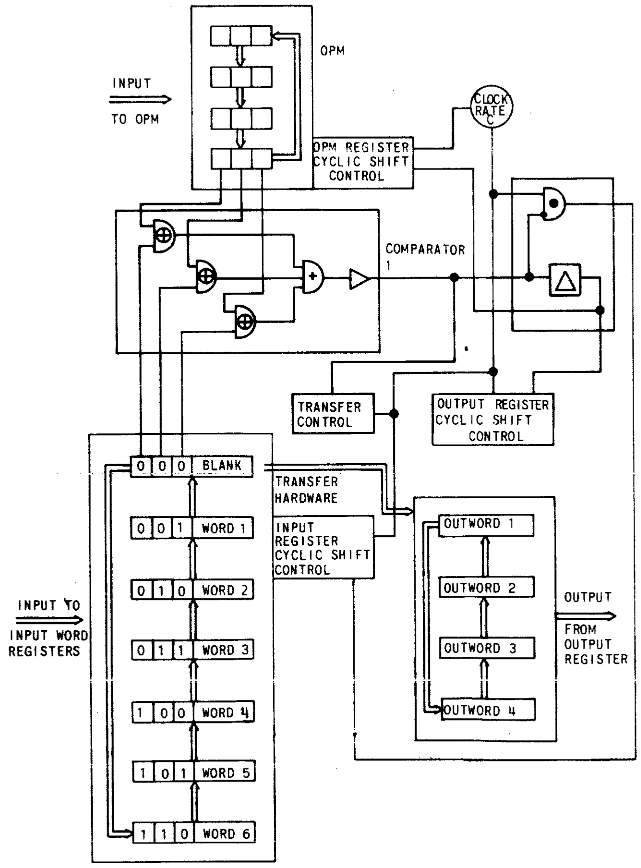


Figure 10—Programmable indexing network

work, the number of input words (n) has no special relation to the number of output words (m). The non-blank output words may appear in many contiguous subsets of the output words (these subsets could be empty). Also, it is desirable to process data during its routing (e.g., matrix transposition in addition to simple sorting or merging).

Thurber proposed two general solutions. The first is essentially a circulating storage device with a map. The device contains an input storage device and an output storage device. The map determines how items are moved from the input to the output. Shift registers were envisioned as being used. The speed of the solution was dependent upon the amount of parallelism built into the network.

If N is a network with n inputs and m outputs then the output position map (OPM) is a vector containing m distinct cells with $\log_2(n+1)$ binary bits per cell. Each cell contains the binary code corresponding to the input value desired in the corresponding output cell. $\text{Log}_2(n+1)$ bits are needed since the n inputs and the 0 (blank) must have a code so that they can be specified as output values if desired.

Figure 10 is an example of a typical indexing network. The input word registers are shifted until the data tag matches the OPM value then the value is transferred and the OPM advanced. Obviously, this process is sequential and could require $n \times m$ shift cycles. Variations on this theme include bidirectional shifting of the registers and increased hardware parallelism.

The second solution was called the "Splitter" and is shown in Figure 11. This solution required that each piece of data be furnished with a map (tag) called an input position map. The input position map (IPM) is a set of binary codes associated with the input data of a network that specifies the position (or positions) that the data is to be transferred to in the set of output registers. The allowable data transfers are indicated in Figure 12. The Splitter operates by transferring (at each stage) the top most piece of data according to the map. Eventually as indicated in Figure 11, the input sequence is reordered. Pipelining is possible through the network. The most difficult and limiting part of these solutions is that the maps must be preconstructed to sort the data based upon the tags.

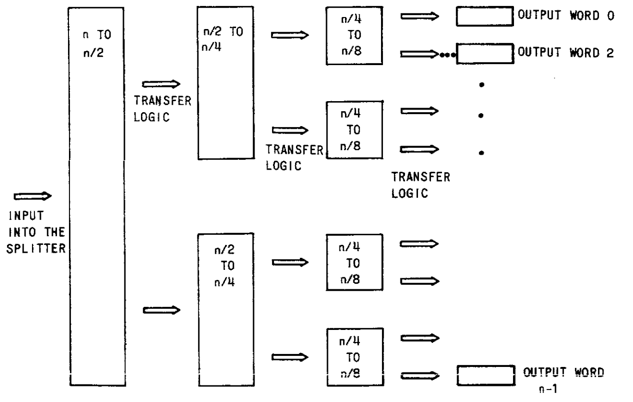


Figure 11—Splitter

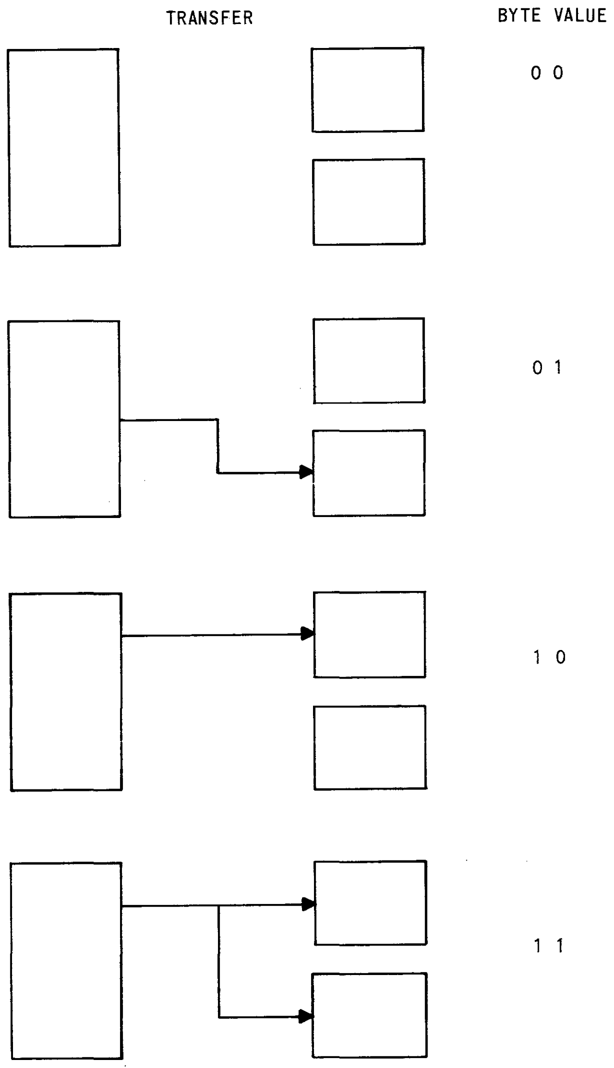


Figure 12—Allowable splitter transfers

Thurber's cascaded permutation network

Thurber[18] studied the problem of constructing a permutation network from a cascade of identical cells such as shown in Figure 13. Figure 14 indicates the implementation of one

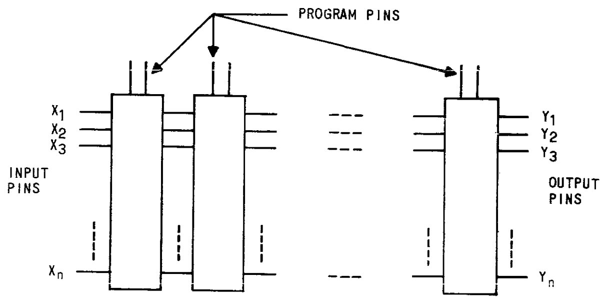
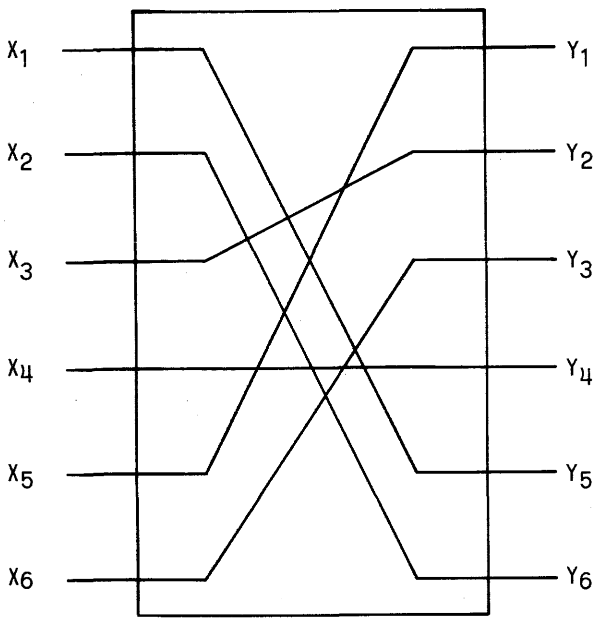


Figure 13—Cascade permuter



$$\begin{pmatrix} X_1X_2X_3X_4X_5X_6 \\ Y_5Y_6Y_2Y_4Y_1Y_3 \end{pmatrix} \sim \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 4 & 1 & 3 \end{pmatrix}$$

Figure 14—Permutation cell

possible permutation and Figure 15 indicates the effect of the composition operator. Given the knowledge of how the basic permutation is realized and composed, the problem then resolves itself into selecting (via permutation group theory) a base which spans the space of all permutations, but which may be economically realized.

Three obvious solutions for the base were rejected. These were:

(1) The set of $n!$ permutations would allow a single cell in the cascade but require many pins and large amounts of logic.

(2) The set of two permutations

$$\begin{pmatrix} 1234\ldots n \\ 2134\ldots n \end{pmatrix} \text{ and } \begin{pmatrix} 1234\ldots n-1\; n \\ 2345\ldots n \quad 1 \end{pmatrix}$$

because this required cascades of length $n!$.

(3) A solution based upon cardinality of permutation orbits because there is no generation algorithm for control.

Solutions selected were:

(1) A base of n elements in which a permutation of n elements is inductively implemented using a permutation of $n-1$ elements and the permutation $\binom{1234\ldots n-1\; n}{2345\ldots n \quad 1}$

(2) A base of $2n-4$ elements in which permutations were added to the base of (1) above to reduce the cascade length.

(3) A base choosen so that the cascade length did not exceed $n-1$ cells. This solution increased the base size to approximately n^2.

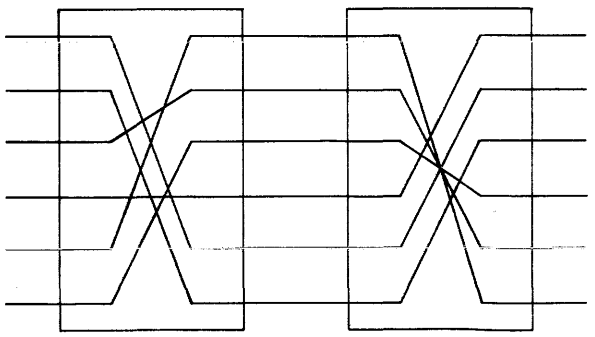
Detailed examples, calculations of length, control algorithms, etc., are given in Reference 18.

Tarjan's sorting network

Tarjan[19] considered the problem of sorting a sequence of numbers using stacks and queues. Three types of basic elements were used. These were queues (FIFO elements), stacks (LIFO elements), and a switchyard. A switchyard is represented by a directed graph with a unique source and sink in which a vertex represents a siding. A siding can be either a queue or stack. Further assumptions include (1) the source and sink sidings must be queues and (2) storage may only occur at vertices of the graph. Tarjan concentrated on determining properties of sorting networks and their relationship to the ability to sort sequences. No specific networks are derived in the paper. The paper presents a good insight into the difficulty of describing general properties of sortable sequences. One important result derived is that if $m \geq 2$, then a sequence of length $3 \cdot 2^{m-1}$ (or less) may be sorted using m stacks in series.

Harada's sequential permutation networks

Harada[20] proposed two permutation networks. The first is a representation of the general permutation $\binom{1\;2\;3\ldots n}{p_1 p_2 p_3 \ldots p_n}$. The second is a lexicographic network representation. Its behavior may be computationally described as an explicit representation of the factorial counting function $N = \sum_{h=2}^{n} A_h \cdot (h-1)! < n!$, where A_h is a h-nary factorial digit and $0 \leq A_h \leq (n-1)$.



$$\begin{pmatrix} 123456 \\ 562413 \end{pmatrix} \begin{pmatrix} 123456 \\ 654123 \end{pmatrix} = \begin{pmatrix} 123456 \\ 235164 \end{pmatrix}$$

Figure 15—Composition operator example

Harada provides a detailed examination of the relationship between the two networks and their control states. His analysis begins with the observation that given the ability to generate all permutations of $k-1$ elements, another state in the network can be constructed (utilizing more elements) which can construct all permutations of k elements. The networks employed are two versions of the Kautz[14] et al. triangular array. As shown in Figure 8, each cell may bend or pass a variable. Two complementary networks (shown in Figures 16 and 17) are defined. The network of Figure 16 works like the triangular array of Kautz. The network of Figure 17 is the lexicographic network.

The goal of the study was to define a network which would produce permutations for computational purposes not an interconnection network, but the network could be used as an interconnection network. The following conclusions were reached about each network:

(1) each could generate all $n!$ permutations
(2) a permutation could be derived from its predecessor in a single clock step.
(3) the permutor of Figure 16 represents the equation $\binom{1\ 2\ 3 \cdots n}{p_1 p_2 p_3 \ldots p_n}$ and is input oriented.
(4) the permuter of Figure 17 represents the factorial counting function and is output oriented.

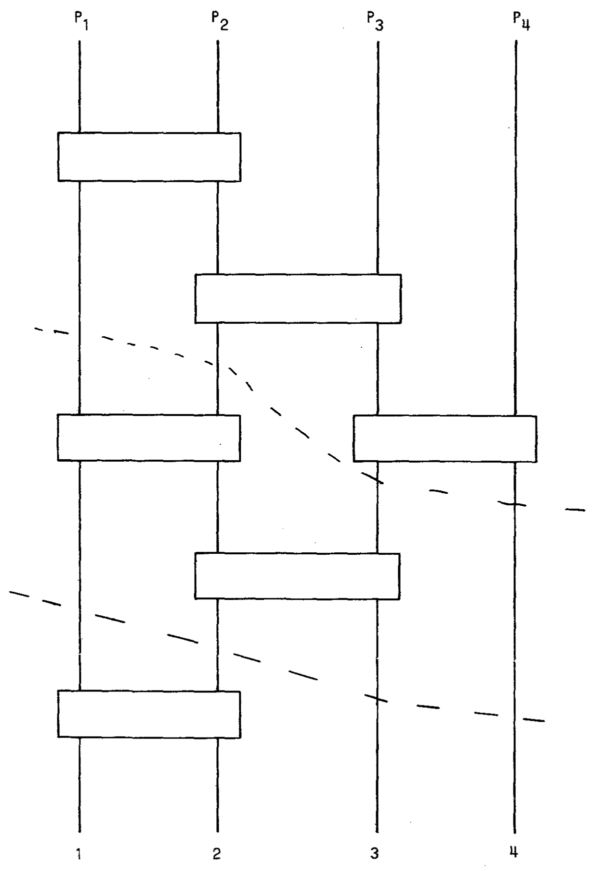


Figure 16—Harada permutation array

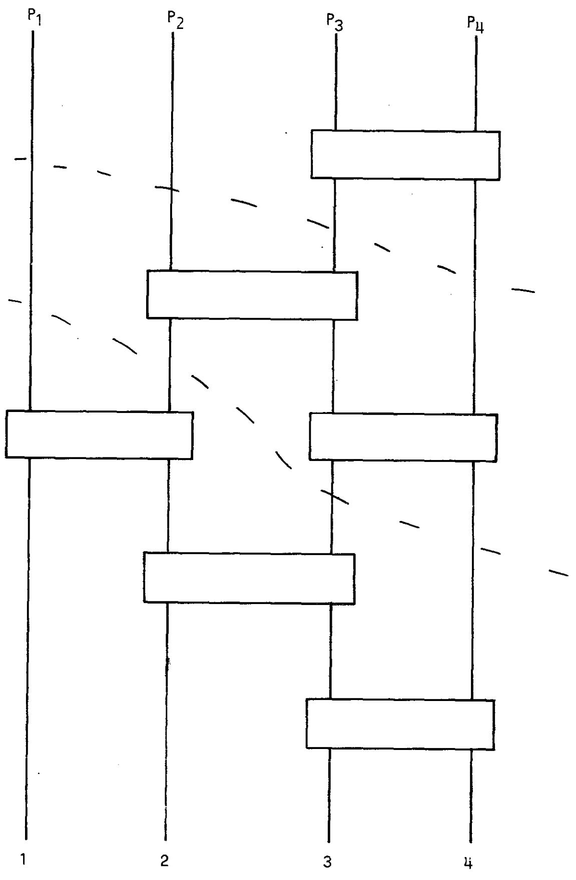


Figure 17—Harada lexicographic permutation array

Bandyopadhyay's permuter network

Bandyopadhyay[21] et al. discussed the implementation of the network shown in Figure 18. The concept was based upon the generation of a permutation of K elements by being able to generate a permutation of $K-1$ elements. The selector cells were realized using cellular logic arrays. The advantage of this approach is the small number of control lines. The disadvantage in comparison to Thurber's[18] cascade approach is that all cells are unique and thus the concept is suitable for LSI implementation only if cellular arrays can be used in the implementation as proposed.[21]

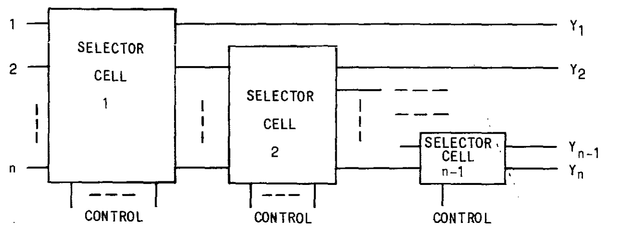


Figure 18—Bandyopadhyay cascade permuter

Smith's sorting network analysis

Smith[22] considers a number of problems concerning the concept of a sorting network. Whereas, most of the other references are mainly interested in how to synthesize sorting networks, Smith was interested in whether a network sorts and the characteristics of networks which sort items.

Smith views the network as a sequential finite state machine. Comparators are finite state machines and, thus, compositions of comparators are finite state machines. Furthermore, viewing the network as a composition of finite state machines, he is able to analyze the number of permutations that may be produced by any one given configuration of comparators (of any mixed variety). A large number of theorems describing the behavior of comparator networks are derived. Several important results are:

(1) No two reachable states in a sorting network are equivalent
(2) A sorting network on an n-element domain has $\sigma(n)=\sum_{k=0}^{n} k!\cdot\left(\begin{smallmatrix}n\\k\end{smallmatrix}\right)$ reachable states where $\left(\begin{smallmatrix}n\\k\end{smallmatrix}\right)$ denotes a Stirling number of the second kind.

The Smith paper also introduces the concept termed the zero-one principle and shows how this principle may be applied to determine whether a network sorts or not. One of the most important results of his investigation is that no comparator network containing one or more comparators can sort each of the assignments

```
000 . . . 1
 . . .    .
 . . .    .
000 . . . 1
001 . . . 1
011 . . . 1
```

without sorting some other assignment. More importantly the paper notes some of the reasons why the problem of designing sorting networks is so difficult: i.e.,

(1) lack of algebraic structure on the subject;
(2) lack of tools to manipulate composite networks of comparators; and
(3) lack of tools to manipulate sets of network assignments,

Smith did, however, discover that it seemed easier to approach the subject from the standpoint of Boolean expressions rather than from the standpoint of partially ordered sets. Furthermore, he did not see any promising opportunities for resolution of even the simplest questions, such as, how many comparators are required to construct a sorting network.

Lawrie's Ω networks

Lawrie[23] developed a set of connection networks and algorithms for their synthesis based upon the concept of the mathematical notion of a "Ω base" representation of integers. The research was being done to determine effective means of interconnecting processing elements in highly parallel processor configurations.

Lawrie shows that if R_n is an ordered set of integer factors of $n (R_n=(p_1, \ldots, p_k)$ such that $p_1 \cdot p_2 \cdot \ldots p_k=n)$ then $\Omega(R_n)$ may be defined as the set $\Omega(R_n)=(W_1/W_k=1, W_i=W_{i+1}\cdot p_{i+1})$ where $0<i\leq k-1$ and $W_i=\prod_{j=i+1}^{k} p_j$. Also note that $W_0=n$ and $W_1/W_k=1$ means that there exists an integer, a, such that $W_1\cdot a=W_k$. Using $\Omega(R_n)$, Lawrie develops a connection network algorithm. The algorithm constructs a k stage network (stages numbered $1, 2, \ldots, k$ from left to right). The ith stage is composed of n/p_i crossbar switches, each switch p_i by p_i. At each stage the n/p_i crossbar switch networks have their inputs and outputs labeled from 0 to $n-1$ (e.g., the inputs and outputs of switch number 2 are $p_i, p_{i+1}, \ldots, 2p_i-1$). Connections are made by connecting output j of stage i to input l of stage $i+1$ where $l=(j\div W_{i-1})\cdot W_{i-1}+(j \bmod p_i)\cdot W_i+(j \bmod W_{i-1})\div p_i$. Where $X\div Y$ is taken to mean the integer part of the quotient. A special set of connections must be computed for stage 1. A control algorithm is developed by constructing a n-set, which represents a mapping or connection between n input and n output nodes. n-sets can be used to describe the control of the network and to detect conflicts in routing. A detailed study of the properties, effectiveness, and construction of Ω networks is given in Reference 23.

One Ω network which Lawrie implemented is shown in Figure 19. This is a bit serial network constructed from building blocks that transmit and/or block signals in response to the strobe or reset at each cell. Transmissions allowed are $A=C$, $B=D$, or $A=D$, $B=C$. Conflicts can occur but may also be resolved in this network scheme.

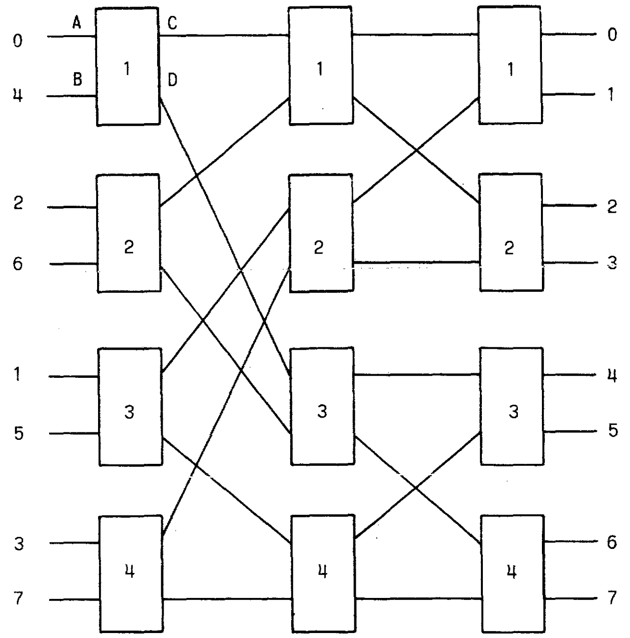


Figure 19—Ω network

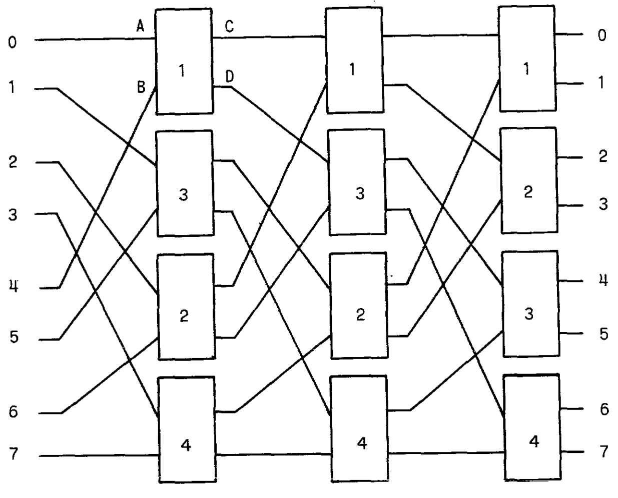


Figure 20—Redrawn Ω network

There are a number of things to note about the Ω network of Figure 19. First, topologically, its last stages represent bitonic sorters and rearrangeable networks. Further when it is redrawn as shown in Figure 20, it contains the perfect shuffle interconnections[26,27] and the interconnections become quite regular.

There are many forms of Ω networks. Lawrie has analyzed numerous variations which minimize various pertinent factors. His work is probably the best analytic study of interconnection networks performed to date. It provides us with the most advanced mathematical tools so far developed in the area of sorting and interconnection networks. Crossbar switches, bitonic sorters, etc., can be viewed as special cases of Ω networks.

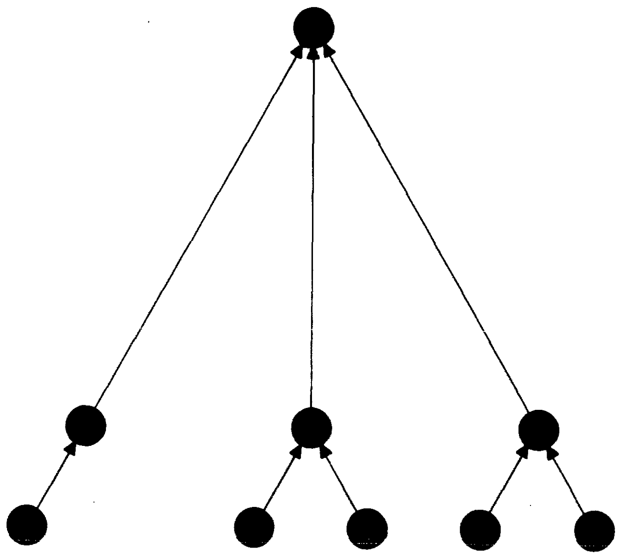


Figure 21—Banyan

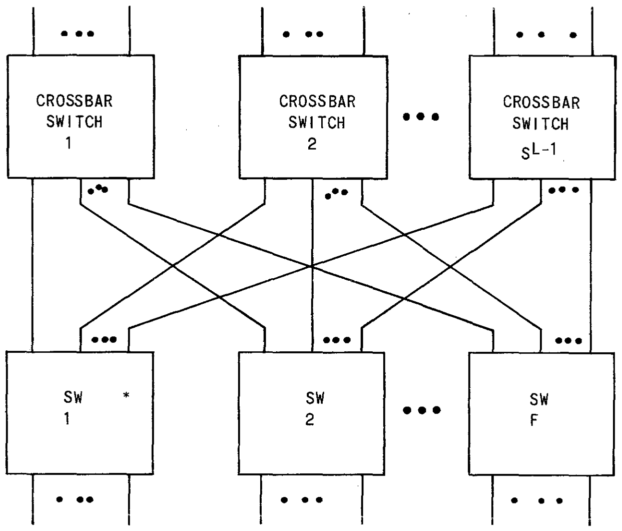


Figure 22—L-level SW banyan structure with fanout F and spread S

Goke and Lipovski's banyan networks

A "banyan" as defined by Webster is an East Indian fig tree whose branches grow shoots that take root and become new trunks. A banyan[24] can be illustrated graphically. A graph of a banyan is a Hasse diagram of a partial ordering in which there is one and only one path from any base to any apex. A base constitutes any vertex with no incident arcs. An apex is any vertex without arcs incident from it. All other vertices are called intermediates. Figure 21 is a banyan. The significance of banyan networks is that their tree shaped structure provides the possibility of allowing low propagation delays with limited fan out systems. Further, priority hardware can be included to minimize conflicts and conflict detection schemes can be devised.

Large banyan networks may be synthesized recursively from smaller networks. Goke and Lipovski prove a number of theorems relating to banyan networks, which will allow synthesis of large banyan networks and determination of the networks potential for conflicts.

L-level banyan (a banyan whose vertices are arranged in levels so that arcs can only exist between vertices in adjacent levels) are considered for propogation delay factors (distance between base and apex is the critical factor), conflicts (a criteria to avoid conflicts and enhance performance is derived), and speed factors.

There are two quite important banyan networks. These are the SW[45] and CC (cylindrical crosshatch) structures. The SW structure recursively expands to a crossbar switch as illustrated in Figure 22. The CC structure is rectangular and contains $N=S^L$ vertices in each level. If $V_K{}^0$, $V_K{}^1$, . . ., $V_K{}^{n-1}$ are the vertices in each level K of an L-level CC banyan structure then there is an arc from a vertex $V_K{}^i$ to a vertex $V_{K+1}{}^j$ in the level above whenever $j=i+mS^K (\text{mod } n)$ for some $m=0, 1, \ldots, S-1$. Furthermore, crossbar switches, bitonic sorters, etc., are able to be described and analyzed as banyans.

TABLE I—Comparison of Interconnection Network Approaches

NETWORK TYPE \ PARAMETERS	DATE TRANSMISSION TIME	NUMBER OF CELLS	TYPE OF CELLS	UPPER BOUND ON CYCLES REQUIRED FOR AN ARBITRARY PERMUTATION	COMMENTS
BARREL SHIFTER	$\mathrm{LOG}_2\ n$	VARIABLE DEPENDING ON CELL CHIPS USED	SELECTOR CHIPS	$\mathrm{LOG}_2\ n$	COMMON ELEMENT IN COMPUTER SYSTEMS
CROSSBAR	$\mathrm{LOG}_2\ n$	n^2	CROSSPOINT	$\mathrm{LOG}_2\ n$	COMMON ELEMENT IN SOME COMPUTER SYSTEMS.
BENES	$2\ \mathrm{LOG}_2\ n$	$n\ \mathrm{LOG}_2(n)-n+1$	TWO INPUT TWO OUTPUT SWITCHES	$2\ \mathrm{LOG}_2\ n$	EARLY BUT WELL STUDIED REARRANGEABLE NETWORK CONCEPT.
BATCHER	$(\mathrm{LOG}_2\ n)^2$	$n/4((\mathrm{LOG}_2\ n)^2-(\mathrm{LOG}_2\ n)+4)$	TWO INPUT TWO OUTPUT SWITCHES	$(\mathrm{LOG}_2\ n)^2$	BITONIC SORTER
KAUTZ	$n-1$	$N^2/2$ TO $n\ \mathrm{LOG}_2 n$	TWO INPUT TWO OUTPUT SWITCHES	$n-1$	MANY DIFFERENT SOLUTIONS TOPOLOGIES
THURBER	nm TO n	$(n+m)(p)$	SHIFT REGISTER CELLS	nm TO n	MANY DIFFERENT SOLUTIONS REQUIRING VARYING AMOUNTS OF CONTROL HARDWABE
THURBER	$n-1$ TO $L_n=n-1+L_{n-1}$ $(L_3=2)$	$n-1$ TO $L_n=n-1+L_{n-1}$ $(L_3=2)$	COMPLEXITY FROM $2.5n^2+.5n$ TO $7n$ GATES PER CELL	$n-1$ TO $L_n=n-1+L_{n-1}$ $(L_3=2)$	CASCADES RUN INTO PIN LIMITATIONS QUICKLY
LAWRIE	$\mathrm{LOG}_2\ n$	$n\ \mathrm{LOG}_2\ n$	VARIES	$\mathrm{LOG}_2\ n$	MATHEMATICALLY SOPHISTICATED ANALYTIC AND SYNTHESIS TOOLS
GOKE	$\mathrm{LOG}_2\ n$	$n\ \mathrm{LOG}_2\ n$		n	GROWTH RATES MAY VARY CONSIDERABLY DEPENDING UPON FAN OUT ASSUMPTIONS

APPLICATIONS

There are a number of applications for which sorting and interconnection networks may be used. Some typical ones are listed below:

(1) sorting data
(2) switching networks with priority (conflict) resolution
(3) multiaccess memories
(4) content addressable memories
(5) recirculating memories
(6) permutation generators
(7) restructurable bussing structures
(8) interconnection networks for parallel processors

Some actual problems requiring (or solved by) application of interconnection networks may be found in References 25-32. Further references of interest pertinent to the subject of interconnection networks begin with Reference 33 and continue to Reference 48.

COMPARISON OF INTERCONNECTION NETWORKS

It is very difficult to make a comparison of interconnection networks for the following reasons:

(1) There is very little theory and analytic tools available.
(2) The networks were mainly developed *ad hoc* and thus satisfy varying requirements, making them even more difficult to compare due to their different capabilities.
(3) We have no baseline networks to reference.
(4) There are really two structures to consider (control and data routing).

Table I has been constructed to give the reader a feel for some global differences in the networks. Presented in this table are some very gross comparisons of the "order-of-magnitude" type. The table is admittedly quite rough, because the detailed comparison of such networks is far beyond the scope of this paper. Barrel shifters and crossbar switches have been included to provide a baseline feel for the networks complexity. Notably, the order-of-magnitude properties of the networks are quite similar so that differences in the networks lie in the areas of ease of control, detailed capabilities, practicality, ease of reduction to design, required number of gates, ability to utilize off-the-shelf parts for construction, number of pins, size, weight, power, etc. Furthermore, data transmission time and the upper bound on number of cycles are identical, except one must note that you need to multiply data transmission time by a time factor T for each stage to account for time delays and to obtain a realistic data transmission time figure. T may be different for each cell type thus producing different speed networks.

CONCLUSION

The most important issue clouding the future of interconnection networks is their practicality. Numerous ad hoc approaches have been developed which demonstrate that interconnection networks can be designed but that reduction of such networks to economically viable units is very difficult. A number of theoretical developments[23,24,46] have been introduced which provide good methodologies for analytically designing interconnection networks. Notably, the complexity, delay time, number of units required, etc. do not differ significantly between approaches. Thus it is time to consider the practical problems associated with interconnection networks. Some of the practical issues that must be resolved before interconnection networks can become viable components of a computer system include:

1. Determination whether any additional functions are appropriate to be included in the networks
2. Design of basic building block modules which provide good gate to pin ratios, are low cost, and provide the ability to be utilized to construct larger networks.
3. Development of cost effective LSI partitioning of the networks and their basic building blocks
4. Development of simple, easily changeable control algorithms
5. Investigation of the problems involved with actually transmitting signals through the networks.

To date work on interconnection networks has generally concentrated on achieving the network functionality and ignored the problems associated with implementing the networks. Economically viable networks have yet to be achieved. Furthermore, many difficult problems remain to be solved before the networks approach practicality. From the view point of the network's functional characteristics, real progress is being made. Some of the recent advances[23,24,26] include a number of early network designs as special cases. Thus a strong analytic base has been laid.

In distributed systems, systems of many small processors, or systems constructed from modular logic, the interconnection logic may be at least as expensive as the logic in the processors themselves, thus the recurring hardware cost of modular computer networks may be two to ten times that of a general purpose sequential machine capable of doing the same job.[48] Therefore, deriving economical interconnection networks and bus structures will be more important than ever due to advances in LSI technology. The theoretic future of interconnection networks appears bright, but the real challenge lies in reducing to practice economical interconnection networks which will make possible digital systems with a high degree of interconnectivity.

REFERENCES

1. Joseph, E. C., "Future Computer Architecture in Polysystems," *Proceedings of COMPCON 72*, pp. 149-153, September 1972.
2. Johnson, M. D., et al., "All Semiconductor Distributed Aerospace Processor/Memory Study; Volume II: Description of Alternatives and Trade Offs," Final Report Air Force Contract F33615-72-C-1709, August 1973.
3. Wulf, W. A., and C. G. Bell, "C.mmp—A Multi-mini-Processor," *Proceedings of the 1972 FJCC*, pp. 765-777, December 1972.
4. Thurber, K. J., et al., "A Systematic Approach to the Design of Digital Bussing Structures," *Proceedings of the 1972 FJCC*, pp. 719-740, December 1972.
5. Martin, W. A., "Sorting," *Computing Surveys*, December 1971, pp. 147-174.
6. Benes, V. E., "Algebraic and Topological Properties of Connecting Networks," *Bell System Technical Journal*, pp. 1249-1273, July 1962.
7. Benes, V. E., "Permutation Groups, Complexes, and Rearrangeable Connecting Networks," *Bell System Technical Journal*, pp. 1619-1640, July 1964.
8. Benes, V. E., "Optimal Rearrangeable Multistage Connecting Networks," *Bell System Technical Journal*, pp. 1641-1656, July 1964.
9. Benes, V. E., *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.
10. Joel, A. E., "On Permutation Switching Networks," *Bell System Technical Journal*, pp. 813-822, May-June 1968.
11. Opferman, D. C. and N. T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks," *Bell Systems Technical Journal*, pp. 1579-1618, May-June 1971.
12. Batcher, K. E., "Sorting Networks and Their Applications," *Proceedings of the 1968 SJCC*, pp. 307-314.
13. Batcher, K. E., "Means for Merging Data," U.S. Patent 3428946, February 18, 1969.
14. Kautz, W. H., et al., "Cellular Interconnection Arrays," *IEEETC* May 1968, pp. 443-451.
15. Waksman, A., "A Permutation Network," *JACM*, January 1968, pp. 159-163.
16. Bose, R. C. and R. J. Nelson, "A Sorting Problem," *JACM*, Sept. 1962, pp. 282-296.
17. Thurber, K. J., "Programmable Indexing Networks," *Proceedings of the 1970 SJCC*, pp. 51-58.
18. Thurber, K. J., "Permutation Switching Networks," *Proceedings of the 1971 Computer Designer's Conference*, January 19-21, 1971, Anaheim, California, pp. 7-24, Industrial and Scientific Conference Management, Chicago, Illinois.
19. Tarjan, R., "Sorting Using Networks of Queues and Stacks," *JACM*, April 1972. pp. 341-346.
20. Harada, K., "Sequential Permutation Networks," *IEEETC*, May 1972, pp. 472-479.
21. Bandyopadhyay, S., et al., "A Cellular Permuter Array," *IEEETC*, October 1972, pp. 1116-1119.
22. Smith, B. J., "An Analysis of Sorting Networks," Final Report ONR Contract N00014-70-A-0362-0006, October 1972.
23. Lawrie, D. H., Memory-Processor Connection Networks, University of Illinois Report UIUCDCS-R-73-557, February 1973.
24. Goke, L. R. and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems," *1st Annual Computer Architecture Conference*, Gainsville, Florida, December 1973, pp. 21-28.
25. Rohrbacher, D. L., Advanced Computer Organization Study: Volume 1—Basic Report, Volume 2—Appendixes, Air Force Contract AF 30(602)-3550, April 1966, AD 631870 and AD 631871.
26. Thurber, K. J. and J. W. Myrna, "System Design of a Cellular APL Machine," *IEEETC*, April 1970, pp. 291-303.
27. Pease, M. C., "An Adaption of the Fast Fourier Transform to Parallel Processing," *JACM*, pp. 252-264, April 1968.
28. Stone, H. S., "Parallel Processing with the Perfect Shuffle," *IEEETC*, pp. 153-161, February 1971.
29. Stone, H. S., "Dynamic Memories with enhanced data access," *IEEETC*, pp. 359-366, April 1972.
30. Gold, D. E., "Applications of Some Switching Network Results to Dynamic Allocation of Memories in a Hierarchy," *COMPCON 72*, pp. 127-129.
31. Kuck, D. J., et al., "Interconnection Networks for Processors and Memories in Large Systems," *COMPCON 72*, pp. 131-134.
32. Wong, C. K. and P. C. Yue, "The Anticipatory Control of a Cyclically Permutable Memory," *IEEETC*, May 1973, pp. 481-488.
33. Liu, C. L., "Construction of Sorting Plans," *Theory of Machines and Computations*, pp. 87-98 (eds. Z. Kohavi and A. Paz) Academic Press, New York, 1971.
34. Van Voorhis, D. C., A Lower Bound for Sorting Networks that use the Divide-Sort-Merge Strategy, Stanford Digital Systems Laboratory Technical Report No. 17, August 1971.
35. Levitt, K. N., et al., "A Study of the Data Communication Problems in a Self-Repairable Multiprocessor," *Proceedings of the 1968 SJCC*, pp. 515-527.
36. Clos, C., "A Study of Non-Blocking Switching Networks," *Bell System Technical Journal*, March 1963, pp. 406-424.
37. Tsao-Wu, N. T., and D. C. Opferman, "On Permutation Algorithms for Rearrangeable Switching Networks," *Conference Record 1969 IEEE International Conference On Communications*, pp. 10.29-10.34.
38. Kautz, W. H. and J. Turner, "Universal Connecting Networks and the Synthesis of Canonical Sequential Circuits," *Proceedings of the 9th Annual Symposium on Switching and Automata Theory*, pp. 257-268, 1968.
39. Elspas, B., "Topological Constraints on Interconnection-limited logic," *Proceedings of the 5th Annual Symposium on Switching Circuit Theory and Logical Design*, pp. 133-137, 1964.
40. Joel, A. E., Relay Permutation Type Switching System, U.S. Patent 2,625,610 January 13, 1953.
41. Moore, E. F., Relay Selecting Circuit, U.S. Patent 2,864,008, December 9, 1958.
42. Kautz, W. H., et al., Cellular Logic-in-Memory Arrays, Final Report ONR Contract NONR-4833(00), May 1970.
43. Waksman, A., "On Permutation Networks," *Proceedings of the Hawaii International Conference on System Sciences*, January 1968, pp. 581-582.
44. Kautz, W. H., "Cellular Logic-in-Memories," *IEEETC*, August 1969, pp. 719-727.
45. Lipovski, G. J., "The Architecture of a Large Associative Processor," *Proceedings of the 1970 SJCC*, pp. 385-396.
46. Kautz, W. H., "The Design of Optimum Interconnection Networks for Multiprocessors," *Structure et Conception des Ordinateurs Architecture and Design of Digital Computers*, (ed. Guy Boulaye), Ecole dete de 1 O.T.A.N. A N.A.T.O. Advanced Summer Institute 1969, Dunob, Paris, 1971.
47. Elspas, B., et al., Theory of Cellular Logic Networks and Machines, Final Report Contract F19628-68-C-0262, 1968.
48. Fuller, S. H., and D. P. Siewiorek, "Some Observations on Semiconductor Technology and the Architecture of Large Digital Modules," *Computer*, October 1973, pp. 15-21.

An economical construction for sorting networks

by DAVID C. VAN VOORHIS

IBM
Los Gatos, California

INTRODUCTION

An N-input *sorting network*, or an N-*sorter*, is a switching circuit with N outputs that satisfy the following: for any combination of inputs $I=\{i_0, i_1, \ldots, i_{N-1}\}$, the resulting outputs $0=\{o_0, o_1, \ldots, o_{N-1}\}$ are a permutation of I, and $o_0 \leq o_1 \leq \cdots \leq o_{N-1}$. Batcher[1] shows that a basic 2-sorter, or *comparator* cell, can be used to construct N-sorters for arbitrary N. For example, the circuit in Figure 1 is a 4-sorter, since comparators A through D move the smallest input to o_0 and the largest input to o_3, and then comparator E orders the remaining two inputs.

The study of sorting networks that use comparators is motivated only in part by the desire to build special hardware for sorting; the following areas of research would also benefit from improved constructions for such circuits:

1. Permutation Networks—An N-input permutation network is simply a switching circuit that performs the function of an $N \times N$ crossbar switch. Waxman[2] shows that a basic 2-permuter cell quite similar to a comparator can be used to construct an N-input permutation network for arbitrary N. Although Waxman's circuit is cheaper and faster by a factor of $\sim\log_2 N$ than the best N-sorter known, the effort required to control (that is, "program") the minimum cell permutation network makes it less attractive than a sorting network for some applications.
2. Nonadaptive Sort Algorithms—For a number of popular internal sort algorithms, such as Shellsort[3] and Quicksort,[4] the basic step is to compare two records and to interchange them if they are out of order. The two algorithms cited are *adaptive* in that the particular sequence of "Compare and Conditionally Interchange (CCI)" operations that are performed depends in part on the original order of the records. It is possible, however, to design sort algorithms that use a fixed sequence of CCI operations, and hence are *nonadaptive*. Such sort algorithms may prove especially attractive for pipeline computers, since a fixed sequence of CCI operations doesn't require any conditional branching.
3. Parallel Sort Algorithms—Sort algorithms that use a sequence of CCI operations are well suited for parallel computers, since successive CCI operations can be executed simultaneously (in different processors) if their operands are distinct.

Most of the previous work with sorting networks that use comparators has been devoted to the problem of determining $S(N)$, the minimum number of comparators required by a network that sorts N inputs. This problem is particularly intriguing because the strongest lower bound known[5] for $S(N)$,

$$N(\log_2 N) + .5N(\log_2(\log_2 N)) + 0(N),$$

grows asymptotically as $N(\log N)$, whereas the strongest upper bound known,[6]

$$.250N(\log_2 N)^2 - .386N(\log_2 N) + 0(N),$$

grows as $N(\log N)^2$. This paper shows that

$$S(N) \leq .250N(\log_2 N)^2 - .395N(\log_2 N) + 0(N), \quad (1)$$

which represents an improvement of $\sim .009N(\log_2 N)$ over the best previous upper bound for $S(N)$.

This paper gives a concise notation for sorting networks that use comparators, followed by a brief description of a network[7,8] that previously provided the strongest upper bound known for $S(N)$. Improvements to this network are shown to lead to (1).

COMPARATOR NETWORKS*

Let $\mathbf{x}=\langle x_0, x_1, \ldots, x_{N-1}\rangle$ represent a sequence of N real numbers and let $R(N)$ represent the set of such sequences; $\mathbf{x}$ is said to be *sorted* if $x_0 \leq x_1 \leq \cdots \leq x_{N-1}$. A single operator, the *comparator operator* $(i:j)$, is defined as follows for $\mathbf{x} \in R(N)$ and $0 \leq i < j < N$.

$$\mathbf{x}(i:j)=\mathbf{y}, \qquad \text{where} \quad \begin{cases} \mathbf{y} \in R(N); \\ y_i = \min[x_i, x_j]; \\ y_j = \max[x_i, x_j]; \\ y_k = x_k, \; i \neq k \neq j. \end{cases} \quad (2)$$

A sequence of one or more comparator operators is termed a *comparator network*; Greek letters will be used to represent

* The notation used here is based on that used by Floyd and Knuth[9] and by Knuth.[10]

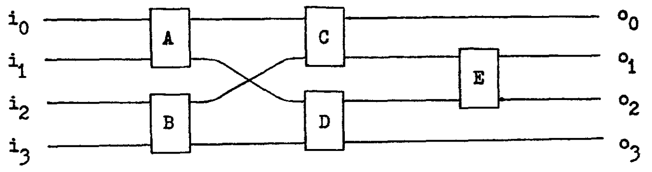


Figure 1—4-sorter

such networks, with $\alpha\beta$ denoting the network consisting of α followed by β. $C(N)$ represents the set of comparator networks whose comparator operators all have the form $(i:j)$ with $j<N$, and a network $\alpha\in C(N)$ is called an N-sorter network if for every sequence $\mathbf{x}\in R(N)$, $\mathbf{x}\alpha$ is sorted. For example, it is easily verified that

$$\alpha=(0:1)(2:3)(0:2)(1:3)(1:2) \qquad (3)$$

is a 4-sorter network.

Our interest in this mathematical formulation of comparator networks is summarized by the following obvious lemma.

Lemma 1:

If α is an N-sorter network that consists of c comparator operators, then:

1. there exists a nonadaptive sort algorithm that includes exactly c CCI operations;
2. there exists an N-input sorting network that contains exactly c comparators; and
3. $S(N)\leq c$.

A sequence $\mathbf{x}=\langle x_0, x_1, \ldots, x_{N-1}\rangle\in R(N)$ is termed a *Boolean sequence* if $x_k\in[0, 1]$ for $0\leq k<N$, and the set of such sequences is denoted $B(N)$. For any sequence $\mathbf{x}\in B(N)$, $|\mathbf{x}|$ and $|\bar{\mathbf{x}}|$ represent, respectively, the number of 1's in $\mathbf{x}$ and the number of 0's in $\mathbf{x}$. The following important theorem, which has been proved independently by several researchers, such as Knuth[10], greatly simplifies the problem of testing a given comparator network $\alpha\in C(N)$ to determine whether it is an N-sorter network.

Theorem 1: (Zero-One Principle)

A comparator network $\alpha\in C(N)$ is an N-sorter network if and only if for every sequence $\mathbf{x}\in B(N)$, $\mathbf{x}\alpha$ is sorted.

Most of the network constructions described in the next two sections are recursive, in the sense that once a particular network $\alpha\in C(N)$ is designed, this design is copied k times in the construction of a larger network $\beta\in C(kN)$. The *substitution operator* $\mathcal{S}(\cdot, \cdot)$ defined as follows permits β to be specified compactly. If $\alpha\in C(N)$, and if $\mathbf{k}=\langle k_0, k_1, \ldots\rangle$ is a (possibly infinite) sorted sequence of nonnegative integers, then $\mathcal{S}(\alpha, \mathbf{k})$ is the comparator network achieved by replacing each comparator operator $(i:j)$ with $(k_i:k_j)$. For example, if

$$\mathbf{k}=\langle m \mid m \text{ is a positive prime}\rangle$$
$$=\langle 1, 2, 3, 5, 7, \ldots\rangle,$$

and if α is given by (3), then

$$\mathcal{S}(\alpha, \mathbf{k})=(1:2)(3:5)(1:3)(2:5)(2:3)$$

The technique used in this paper to design especially efficient networks is first to prove that a particular network $\alpha=\beta\gamma$ is an N-sorter network, and then to show that $\beta\hat{\gamma}$ is also an N-sorter network, where $\hat{\gamma}$ is achieved by removing one or more comparator operators from γ. This pruning is indicated by the *deletion operator* $\mathfrak{D}(\cdot, \cdot, \cdot)$, which is defined as follows. If $\alpha\in C(N)$, and if k and l satisfy $0\leq k, l<N$, then $\mathfrak{D}(\alpha, k, l)$ is the comparator network achieved by removing from α each comparator operator of the form $(i:j)$ where $i<k$ or $j\geq N-l$. For example, if α is given by (3), then

$$\mathfrak{D}(\alpha, 1, 1)=(1:2).$$

THE [g, d] NETWORK STRATEGY

The most economical general strategy known for constructing N-sorter networks, the $[g, d]$ strategy,[7,8] is based on g-way merging. This section briefly describes the $[g, d]$ strategy for the special case that $N=gd$, and the next section adds the further restriction that $g*d=2^r$. The following simple observation shows that, despite these restrictions, the network constructions described can be used to achieve an N-sorter network for arbitrary N.

Lemma 2: (Knuth;[10] Green[11])

If α is an N-sorter network, where $N>1$, then $\mathfrak{D}(\alpha, 0, 1)$ is an $(N-1)$-sorter network.

The description of the $[g, d]$ (gd)-sorter network depends on the observation that any sequence $\mathbf{x}\in B(gd)$ can be treated as a $g\times d$ array whose ith row and jth column are defined as follows* for $0\leq i<g$ and $0\leq j<d$.

$$\text{ROW}(\mathbf{x}, d, i)=\langle x_k\in\mathbf{x} \mid (k/d)=i\rangle;$$
$$\text{COL}(\mathbf{x}, d, j)=\langle x_k\in\mathbf{x} \mid (k//d)=j\rangle.$$

(Note that $\text{ROW}(\mathbf{x}, d, i)\in B(g)$ and $\text{COL}(\mathbf{x}, d, j)\in B(d)$ when $\mathbf{x}\in B(gd)$.)

The $[g, d]$ (gd)-sorter network has the form $\alpha\beta\gamma$, where for any sequence $\mathbf{x}\in B(gd)$,

1. $\text{ROW}(\mathbf{x}\alpha, d, i)$ is sorted, $0\leq i<g$;
2. $\text{COL}(\mathbf{x}\alpha\beta, d, j)$ is sorted, $0\leq j<d$; and
3. $\mathbf{x}\alpha\beta\gamma$ is sorted.

The comparator networks α and β are defined as follows,

* Here "k/d" and "k//d" represent, respectively, the quotient and the remainder that result from the integer division of k by d.

where α_1 is any d-sorter network and β_1 is any g-sorter network.

$$\alpha=\alpha_1\mathcal{S}(\alpha_1, \langle k \mid (k/d)=1\rangle)\ldots\mathcal{S}(\alpha_1, \langle k \mid (k/d)=g-1\rangle); \quad (4)$$

$$\beta=\mathcal{S}(\beta_1, \langle k \mid (k//d)=0\rangle)\ldots\mathcal{S}(\beta_1, \langle k \mid (k//d)=d-1\rangle). \quad (5)$$

Theorem 2 and Corollary 2.1 below lead to a precise definition of γ.

Theorem 2: (Gale and Karp[12])

If the rows of a two-dimensional array are sorted, then sorting the columns of that array preserves the order of the rows.

Corollary 2.1:

If $\mathbf{x}$ is any sequence in $B(gd)$, and if α and β are given by (4) and (5), then ROW$(\mathbf{x}\alpha\beta, d, i)$ and COL$(\mathbf{x}\alpha\beta, d, j)$ are sorted, for $0\leq i<g$ and $0\leq j<d$.

Corollary 2.1 suggests the following definition for γ, the final component of the $[g, d]$ (gd)-sorter network. A comparator network $\gamma\in C(gd)$ is termed a $[g, d]$ *f-network*, where $g, d\geq 2$, if $\mathbf{z}\gamma$ is sorted whenever $\mathbf{z}\in B(gd)$ and ROW$(\mathbf{z}, d, i)$ and COL$(\mathbf{z}, d, j)$ are sorted, for $0\leq i<g$ and $0\leq j<d$. With this definition, we can summarize the $[g, d]$ strategy as follows.

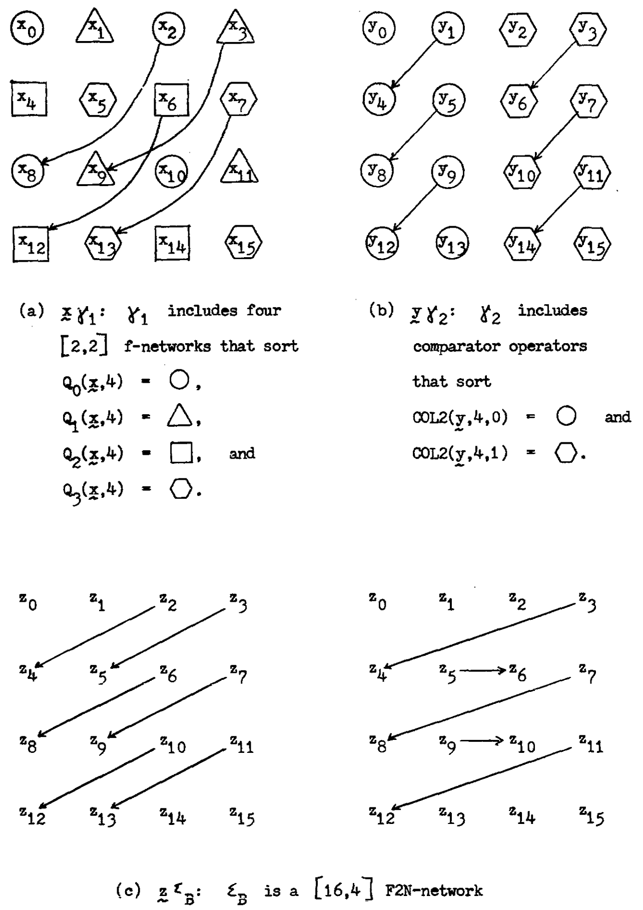

x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15
(a) $x\gamma_1$: γ_1 includes four [2,2] f-networks that sort
$Q_0(x,4)$ = ○,
$Q_1(x,4)$ = △,
$Q_2(x,4)$ = □, and
$Q_3(x,4)$ = ⬡.
y0 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15
(b) $y\gamma_2$: γ_2 includes comparator operators that sort
COL2(y,4,0) = ○ and
COL2(y,4,1) = ⬡.
z0 z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 z12 z13 z14 z15
(c) $z\varepsilon_B$: ε_B is a [16,4] F2N-network

Figure 2—[4,4] f-network

Construction 1: ($[g, d]$ strategy)

If α_1, β_1, and γ are, respectively, a d-sorter network, a g-sorter network, and a $[g, d]$ f-network, where $g, d\geq 2$, and if α and β are given by (4) and (5), then $\alpha\beta\gamma$ is a (gd)-sorter network.

Since the $[g, d]$ strategy is recursive, it can be used to achieve a (4^r)-sorter network for arbitrary r, provided a $[2^r, 2^r]$ f-network can be designed. It is readily verified that the 4-sorter network defined by (3) uses the $[g, d]$ strategy with $g=d=2$, and that

$$\gamma=(1{:}2) \quad (6)$$

is a $[2, 2]$ f-network. The next section describes a recursive construction that yields $[2^r, 2^r]$ f-networks for $r\geq 2$.

$[2^r, 2^r]$ f-NETWORKS

The description of the recursive construction for $[2^r, 2^r]$ f-networks involves the following sequences, which are defined for $\mathbf{x}\in B(2^n)$ and $1\leq r<n$.

$$Q_0(\mathbf{x}, 2^r)=\langle x_k\in\mathbf{x} \mid (k//2)=0,\ ((k/2^r)//2)=0\rangle;$$

$$Q_1(\mathbf{x}, 2^r)=\langle x_k\in\mathbf{x} \mid (k//2)=1,\ ((k/2^r)//2)=0\rangle;$$

$$Q_2(\mathbf{x}, 2^r)=\langle x_k\in\mathbf{x} \mid (k//2)=0,\ ((k/2^r)//2)=1\rangle;$$

$$Q_3(\mathbf{x}, 2^r)=\langle x_k\in\mathbf{x} \mid (k//2)=1,\ ((k/2^r)//2)=1\rangle;$$

$$\text{COL2}(\mathbf{x}, 2^r, j)=\langle x_k\in\mathbf{x} \mid ((k//2^r)/2)=j\rangle, \qquad 0\leq j<2^{r-1}.$$

When $\mathbf{x}$ is considered a $2^{n-r}\times 2^r$ array, these sequences have the following simple interpretation. (See Figure 2(a).)

$Q_0(\mathbf{x}, 2^r)=\langle$even elements on even rows of $\mathbf{x}\rangle$;
$Q_1(\mathbf{x}, 2^r)=\langle$odd elements on even rows of $\mathbf{x}\rangle$;
$Q_2(\mathbf{x}, 2^r)=\langle$even elements on odd rows of $\mathbf{x}\rangle$;
$Q_3(\mathbf{x}, 2^r)=\langle$odd elements on odd rows of $\mathbf{x}\rangle$;
COL2$(\mathbf{x}, 2^r, j)=\langle$elements of COL$(\mathbf{x}, 2^r, 2j)$ and COL$(\mathbf{x}, 2^r, 2j+1)\rangle$.

When $r\geq 2$, the $[2^r, 2^r]$ f-network has the form $\gamma_1\gamma_2\delta$, where if $\mathbf{x}\in B(4^r)$, and if ROW$(\mathbf{x}, 2^r, i)$ and COL$(\mathbf{x}, 2^r, j)$ are sorted, $0\leq i, j<2^r$:

1. $Q_k(\mathbf{x}\gamma_1, 2^r)$ is sorted, $0\leq k\leq 3$;
2. COL2$(\mathbf{x}\gamma_1\gamma_2, 2^r, j)$ is sorted, $0\leq j<2^{r-1}$; and
3. $\mathbf{x}\gamma_1\gamma_2\delta$ is sorted.

The comparator networks γ_1 and γ_2 are defined as follows, where γ is any $[2^{r-1}, 2^{r-1}]$ f-network.

$$\begin{aligned}\gamma_1 = {} & \mathcal{S}(\gamma, \langle k \mid (k//2)=0, ((k/2^r)/2)=0\rangle)\\ & \mathcal{S}(\gamma, \langle k \mid (k//2)=1, ((k/2^r)/2)=0\rangle)\\ & \mathcal{S}(\gamma, \langle k \mid (k//2)=0, ((k/2^r)/2)=1\rangle)\\ & \mathcal{S}(\gamma, \langle k \mid (k//2)=1, ((k/2^r)/2)=1\rangle); \qquad (7)\end{aligned}$$

$$\gamma_2 = (1{:}2^r)(3{:}2^r+2)\ldots(4^r-2^r-1{:}4^r-2). \qquad (8)$$

Theorem 3 below shows that γ_2 does indeed leave COL2$(\mathbf{x}\gamma_1\gamma_2, 2^r, j)$ sorted, $0 \leq j < 2^{r-1}$; also, Theorem 3 leads to a precise definition of δ.

Theorem 3: (Van Voorhis[13])

Let $\mathbf{x}$ be any sequence in $B(4^r)$ for which ROW$(\mathbf{x}, 2^r, i)$ and COL$(\mathbf{x}, 2^r, j)$ are sorted, $0 \leq i, j < 2^r$, where $r \geq 2$. If γ_1 and γ_2 are given by (7) and (8), and if $\mathbf{z} = \mathbf{x}\gamma_1\gamma_2$, then $Q_k(\mathbf{z}, 2^r)$ and COL2$(\mathbf{z}, 2^r, j)$ are sorted for $0 \leq k \leq 3$ and $0 \leq j < 2^{r-1}$, and the distribution of 0's and 1's in $\mathbf{z}$ satisfies:

$$|\overline{Q_0(\mathbf{z}, 2^r)}| \leq |\overline{Q_1(\mathbf{z}, 2^r)}| + |\overline{Q_2(\mathbf{z}, 2^r)}| + 1; \qquad (9)$$

$$|Q_3(\mathbf{z}, 2^r)| \leq |Q_1(\mathbf{z}, 2^r)| + |Q_2(\mathbf{z}, 2^r)| + 1; \qquad (10)$$

$$|\overline{Q_3(\mathbf{z}, 2^r)}| = 0 \Rightarrow |\overline{Q_0(\mathbf{z}, 2^r)}| \geq |\overline{Q_1(\mathbf{z}, 2^r)}| + |\overline{Q_2(\mathbf{z}, 2^r)}| - 1; \qquad (11)$$

$$|\overline{Q_3(\mathbf{z}, 2^r)}| > 1 \Rightarrow |\overline{Q_0(\mathbf{z}, 2^r)}| \leq |\overline{Q_1(\mathbf{z}, 2^r)}| + |\overline{Q_2(\mathbf{z}, 2^r)}|; \qquad (12)$$

$$|Q_0(\mathbf{z}, 2^r)| = 0 \Rightarrow |Q_3(\mathbf{z}, 2^r)| \geq |Q_1(\mathbf{z}, 2^r)| + |Q_2(\mathbf{z}, 2^r)| - 1; \qquad (13)$$

$$|Q_0(\mathbf{z}, 2^r)| > 1 \Rightarrow |Q_3(\mathbf{z}, 2^r)| \leq |Q_1(\mathbf{z}, 2^r)| + |Q_2(\mathbf{z}, 2^r)|. \qquad (14)$$

Constructions 3 through 6 below involve four special comparator networks that are defined as follows. The comparator networks ϵ_B, ϵ_L, ϵ_H, and $\epsilon_N \in C(2^n)$ are termed, respectively, a *$[2^n, 2^r]$ F2B-network*, a *$[2^n, 2^r]$ F2L-network*, a *$[2^n, 2^r]$ F2H-network*, and a *$[2^n, 2^r]$ F2N-network*, where $2 \leq r < n$, if $\mathbf{z}\epsilon_B$, $\mathbf{z}\epsilon_L$, $\mathbf{z}\epsilon_H$, and $\mathbf{z}\epsilon_N$ are sorted whenever

1. $\mathbf{z} \in B(2^n)$,
2. $Q_k(\mathbf{z}, 2^r)$ and COL2$(\mathbf{z}, 2^r, j)$ are sorted for $0 \leq k \leq 3$ and $0 \leq j < 2^{r-1}$, and
3. $\mathbf{z}$ satisfies, respectively: (9)-(14); (9)-(12); (9)-(10) and (13)-(14); and (9)-(10) only.

These definitions imply the following lemma.

Lemma 3

If δ is a $[2^n, 2^r]$ F2N-network, where $2 \leq r < n$, then δ is also a $[2^n, 2^r]$ F2B-network, a $[2^n, 2^r]$ F2L-network, and a $[2^n, 2^r]$ F2H-network.

With the above definitions, we can summarize the $[2^r, 2^r]$ f-network design as follows.

Construction 2:

If γ and ϵ_B are, respectively, a $[2^{r-1}, 2^{r-1}]$ f-network and a $[4^r, 2^r]$ F2B-network, where $r \geq 2$, and if γ_1 and γ_2 are given by (7) and (8), then $\gamma_1\gamma_2\epsilon_B$ is a $[2^r, 2^r]$ f-network.

This construction is illustrated in Figure 2 for the case $r=2$: Figure 2(a) shows $\mathbf{x}\gamma_1$ (with $\mathbf{x}(i{:}j)$ represented by an arrow from x_i to x_j), where γ_1 is achieved by using (6) in (7); Figure 2(b) shows $\mathbf{y}\gamma_2 = \mathbf{x}\gamma_1\gamma_2$, where γ_2 is given by (8); and Figure 2(c) shows $\mathbf{z}\epsilon_B = \mathbf{x}\gamma_1\gamma_2\epsilon_B$, where ϵ_B is the [16, 4] F2B-network described by Construction 3 below (and Lemma 3). Construction 4 below provides a recursive design for F2N-networks, which is illustrated in Figure 3. (Constructions 3 and 4 are justified by Van Voorhis[13])

Construction 3:

The comparator network

$$\begin{aligned}\delta = {} & (2{:}4)(6{:}8)\ldots(2^n-6{:}2^n-4)\\ & (3{:}5)(7{:}9)\ldots(2^n-5{:}2^n-3)\\ & (3{:}4)(5{:}6)\ldots(2^n-5{:}2^n-4), \qquad (15)\end{aligned}$$

where $n \geq 3$, is a $[2^n, 4]$ F2N-network.

Construction 4:

If ϵ_N is a $[2^{n-1}, 2^{r-1}]$ F2N-network, where $3 \leq r < n$, if δ is given by (15), and if

$$\mathbf{i0} = \langle k \mid ((k//4)/2) = 0\rangle, \qquad (16)$$

$$\mathbf{i1} = \langle k \mid ((k//4)/2) = 1\rangle, \qquad (17)$$

then

$$\mathcal{S}(\epsilon_N, \mathbf{i0})\,\mathcal{S}(\epsilon_N, \mathbf{i1})\,\delta$$

is a $[2^n, 2^r]$ F2N-network.

In view of Lemma 3, the $[4^r, 2^r]$ F2N-network described by Constructions 3 and 4 can serve as the $[4^r, 2^r]$ F2B-network required by Construction 2. (The resulting $[2^r, 2^r]$ f-network is precisely that considered by Van Voorhis[8].) Although the $[2^n, 4]$ F2N-network δ given by (15) is the best $[2^n, 4]$ F2B-network known, Constructions 5 and 6 below show that when $r \geq 3$, a $[2^n, 2^r]$ F2B-network requires (at most) $2(r-1)$ fewer comparator operators than the $[2^n, 2^r]$ F2N-network described by Construction 4. Also, Constructions 5 and 6 show that when $r \geq 3$, a $[2^n, 2^r]$ F2L-network and a $[2^n, 2^r]$ F2H-network require (at most) $r-1$ fewer comparator operators than the $[2^n, 2^r]$ F2N-network described by Construction 4. (These constructions are justified by Van Voorhis.[13])

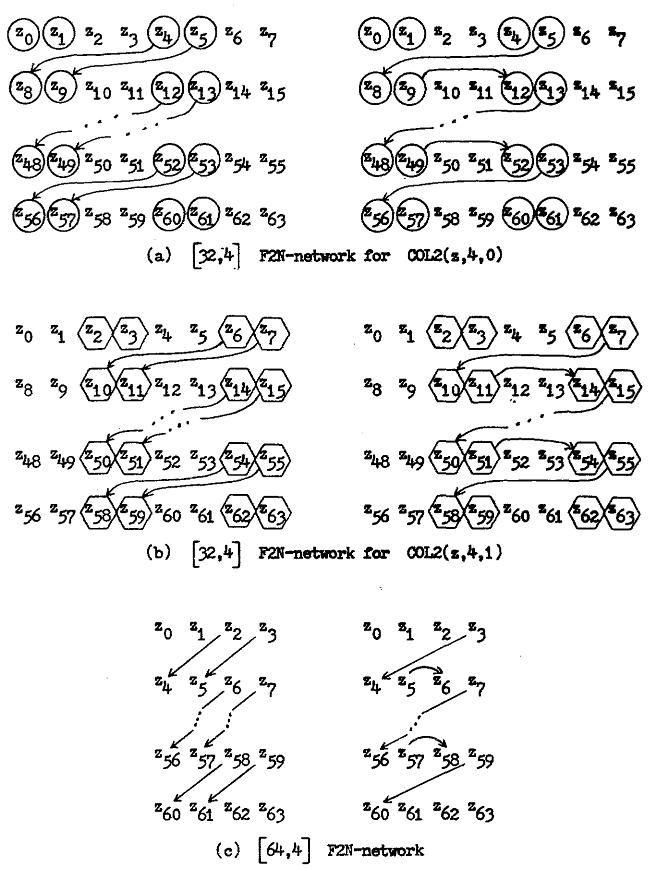

(a) [32,4] F2N-network for COL2(z,4,0)
(b) [32,4] F2N-network for COL2(z,4,1)
(c) [64,4] F2N-network

Figure 3—[64,8] F2N-network

Construction 5:

Let ϵ_L, ϵ_H, and ϵ_N be, respectively, a $[2^{n-1}, 2^{r-1}]$ F2L-network, a $[2^{n-1}, 2^{r-1}]$ F2H-network, and a $[2^{n-1}, 2^{r-1}]$ F2N-network, where $3 \leq r < n$, and let δ, **i0**, **i1**, and α be given by (15)-(17) and

$$\alpha = (2{:}2^r)(3{:}2^r+1)(3{:}2^r)(2^n-2^r-2{:}2^n-4)$$
$$(2^n-2^r-1{:}2^n-3)(2^n-2^r-1{:}2^n-4). \qquad (18)$$

Then

$$\alpha\mathcal{S}(\epsilon_L, \mathbf{i0})\mathcal{S}(\epsilon_H, \mathbf{i1})\mathfrak{D}(\delta, 6, 6),$$

$$\alpha\mathcal{S}(\epsilon_L, \mathbf{i0})\mathcal{S}(\epsilon_N, \mathbf{i1})\mathfrak{D}(\delta, 6, 4),$$

$$\alpha\mathcal{S}(\epsilon_N, \mathbf{i0})\mathcal{S}(\epsilon_H, \mathbf{i1})\mathfrak{D}(\delta, 4, 6),$$

and

$$\alpha\mathcal{S}(\epsilon_N, \mathbf{i0})\mathcal{S}(\epsilon_N, \mathbf{i1})\mathfrak{D}(\delta, 4, 4)$$

are, respectively, a $[2^n, 2^r]$ F2B-network, a $[2^n, 2^r]$ F2L-network, a $[2^n, 2^r]$ F2H-network, and a $[2^n, 2^r]$ F2N-network.

It is readily verified that when

$$\epsilon_L = \epsilon_H = \epsilon_N = \delta,$$

where δ is given by (15), the [64, 8] F2B-, F2L-, F2H-, and F2N-networks described by Construction 5 are all identical to the [64, 8] F2N-network in Figure 3 except that:

1. the networks described by Construction 5 all begin with the six extra comparator operators in α;
2. the networks described by Construction 5 do not require the six comparator operators (2:4), (3:5), (58:60), (59:61), (3:4), and (59:60) in Figure 3(c);
3. the F2B- and F2L-networks do not require the comparator operator (5:6) in Figure 3(c); and
4. the F2B- and F2H-networks do not require the comparator operator (57:58) in Figure 3(c).

Construction 6 below shows further that:

5. the [64, 8] F2B- and F2L-networks described above do not require the comparator operator (9:12) in Figure 3(a); and
6. the [64, 8] F2B- and F2H-networks described above do not require the comparator operator (51:54) in Figure 3(b).

Construction 6:

Let δ, **i0**, **i1**, and α be defined by (15)-(18), where $3 = r < n$, let δ_1 be the $[2^{n-1}, 4]$ F2N-network achieved by replacing 2^n with 2^{n-1} in (15), and let α_1 be achieved by replacing 2^r with 2^{r-1} and 2^n with 2^{n-1} in (18). Then*

$$\alpha\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 6, 4), \mathbf{i0})\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 4, 6), \mathbf{i1})\mathfrak{D}(\delta, 6, 6),$$

$$\alpha\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 6, 4), \mathbf{i0})\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 4, 4), \mathbf{i1})\mathfrak{D}(\delta, 6, 4),$$

$$\alpha\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 4, 4), \mathbf{i0})\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 4, 6), \mathbf{i1})\mathfrak{D}(\delta, 4, 6),$$

and

$$\alpha\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 4, 4), \mathbf{i0})\mathcal{S}(\alpha_1\mathfrak{D}(\delta_1, 4, 4), \mathbf{i1})\mathfrak{D}(\delta, 4, 4)$$

are, respectively, a $[2^n, 8]$ F2B-network, a $[2^n, 8]$ F2L-network, a $[2^n, 8]$ F2H-network, and a $[2^n, 8]$ F2N-network.

NUMERICAL RESULTS

The (4^r)-sorter network described by Construction 1, with $g = d = 2^r$ and $r \geq 2$, shows that

$$S(4^r) \leq 2^{r+1} \cdot S(2^r) + f(2^r, 2^r), \qquad (19)$$

where $f(2^r, 2^r)$ is the minimum number of comparator operators required by a $[2^r, 2^r]$ f-network. The [2, 2] f-network (6) and the $[2^r, 2^r]$ f-network described by Construction 2

* Note that $\alpha_1\mathfrak{D}(\delta_1,6,4)$ is identical to the network achieved by removing the single comparator operator (5:6) from δ_1; similarly, $\alpha_1\mathfrak{D}(\delta_1,4,6)$ can be achieved by removing the single comparator operator $(2^{n-1}-7{:}2^{n-1}-6)$ from δ_1, and $\alpha_1\mathfrak{D}(\delta_1,4,4) = \delta_1$.

show that

$$f(2,2)\le 1, \qquad (20)$$

$$f(2^r,2^r)\le 4\cdot f(2^{r-1},2^{r-1})+\tfrac{1}{2}(4^r-2^r)+\mathrm{F2B}(4^r,2^r),\qquad r\ge 2, \qquad (21)$$

where $\mathrm{F2B}(2^n, 2^r)$ is the minimum number of comparator operators required by a $[2^n, 2^r]$ F2B-network. With $\mathrm{F2L}(2^n, 2^r)$, $\mathrm{F2H}(2^n, 2^r)$, and $\mathrm{F2N}(2^n, 2^r)$ defined similarly, the networks described by Constructions 3 through 6 show that

$$\mathrm{F2N}(2^n,4)\le 2^n-5,\qquad n\ge 3; \qquad (22)$$

$$\mathrm{F2N}(2^n,2^r)\le 2\cdot\mathrm{F2N}(2^{n-1},2^{r-1})+\mathrm{F2N}(2^n,4),\qquad 3\le r<n; \qquad (23)$$

$$\mathrm{F2B}(2^n,8)\le 2^{n+1}-19,\qquad n\ge 4; \qquad (24)$$

$$\mathrm{F2L}(2^n,8)\le 2^{n+1}-17,\qquad n\ge 4; \qquad (25)$$

$$\mathrm{F2B}(2^n,2^r)\le 2\cdot\mathrm{F2L}(2^{n-1},2^{r-1})+\mathrm{F2N}(2^n,4)-2,\qquad 4\le r<n; \qquad (26)$$

$$\mathrm{F2L}(2^n,2^r)\le \mathrm{F2L}(2^{n-1},2^{r-1})+\mathrm{F2N}(2^{n-1},2^{r-1})+\mathrm{F2N}(2^n,4)-1,\qquad 4\le r<n. \qquad (27)$$

(Relation (26) uses the result $\mathrm{F2L}(2^n, 2^r)=\mathrm{F2H}(2^n, 2^r)$, which follows by symmetry.)

Relations (20)-(27) can be used to show that

$$\mathrm{F2N}(2^n,2^r)\le (r-1)2^n-5\cdot 2^{r-1}+5,\qquad 2\le r<n;$$

$$\mathrm{F2L}(2^n,2^r)\le (r-1)2^n-5\cdot 2^{r-1}-r+6,\qquad 3\le r<n;$$

$$\mathrm{F2B}(2^n,2^r)\le (r-1)2^n-5\cdot 2^{r-1}-2r+7,\qquad 3\le r<n;$$

$$f(2^r,2^r)\le \frac{1}{2}r^2 4^r-\left(\frac{103}{72}\right)4^r+3\cdot 2^r+\left(\frac{2}{3}\right)r-\left(\frac{13}{9}\right),\qquad r\ge 2.$$

Therefore, (19) leads to

$$S(4^{2^k})\le\left(4^k+(p+q_k)2^k+\frac{103}{72}\right)4^{2^k}-\frac{3}{2},\qquad k\ge 0,$$

where p is a constant determined by the boundary condition, and

$$q_k=\sum_{i=1}^{k}(12\cdot 2^i+1)/(18\cdot 2^i\cdot 4^{2^i}).$$

The boundary condition $S(4)\le 5$ provided by (3) leads to $p=-29/36$, whereas the boundary condition $S(16)\le 60$ provided by Green[11] leads to $p=-241/288$. Since q_k converges rapidly to .046, the latter value for p shows that

$$S(4^r)\le r^2 4^r-.791r4^r+0(4^r),$$

which is equivalent to (1).

CONCLUSIONS

This paper has considered a refinement of the $[g, d]$ strategy for sorting networks that leads to slightly more economical networks than those previously described. The question arises whether further refinements will lead to even greater savings. The answer is probably "yes," and both of the following approaches seem promising.

1. Strengthen Theorem 3. It can be shown that for $k\ge 1$,
$$|\overline{Q_3(\mathbf{z},2^r)}|<\tfrac{1}{2}k(k+1)\Rightarrow|\overline{Q_1(\mathbf{z},2^r)}|+|\overline{Q_2(\mathbf{z},2^r)}|-k \le|\overline{Q_0(\mathbf{z},2^r)}|+|\overline{Q_3(\mathbf{z},2^r)}| \le|\overline{Q_1(\mathbf{z},2^r)}|+|\overline{Q_2(\mathbf{z},2^r)}|+k,$$
and that a similar relation applies for the 1's in $\mathbf{z}$. These two relations imply (9)-(14), which lead to the savings in Constructions 5 and 6; perhaps other consequences of these relations will lead to further savings.
2. Generalize the $[g, d]$ strategy to higher dimensions. The most economical 18-sorter network known[7] begins by sorting each dimension of a $2\times 3\times 3$ array, and the most economical 16-sorter network known[11] begins by sorting each dimension of a $2\times 2\times 2\times 2$ array. Perhaps a 3- or 4-dimensional version of Construction 2 will yield economical (8^r)-sorter networks or economical (16^r)-sorter networks.

On the other hand, the $[g, d]$ network strategy and its multidimensional extensions are based on g-way merging, and it has been shown[14] that any strategy based on merging requires order $N(\log N)^2$ comparators. Since the strongest lower bound known for $S(N)$ is order $N(\log N)$, future research on sorting networks might be more profitably devoted to:

1. developing a new general strategy for sorting networks; or
2. proving a stronger lower bound for $S(N)$.

(Most researchers feel that the latter task is easier.)

Finally, it should be noted that since hardware costs are expected to diminish significantly in the near future, it may become more desirable to minimize the delay of a sorting network, rather than the number of comparators.

REFERENCES

1. Batcher, K. E., "Sorting Networks and their Applications," *Proc. SJCC*, 1968, pp. 307-314.
2. Waksman, A., "A Permutation Network," *JACM*, Vol. 15, No. 1, Jan. 1959, pp. 159-163.
3. Shell, D. L., "A High Speed Sorting Procedure," *CACM*, Vol. 2, No. 7, July 1959, pp. 30-32.
4. Hoare, C. A. R., "Quicksort," *Comp. J.*, Vol. 5, 1962, pp. 10-15.
5. Van Voorhis, D. C., "Toward a Lower Bound for Sorting Networks," in *Complexity of Computer Computations*, Plenum Press, 1972, pp. 119-129.
6. Drysdale, R. L., *Sorting Networks which Generalize Batcher's Odd-even Merge*, Honors Paper, Knox College, Galesburg, Illinois, May 1973.
7. Van Voorhis, D. C., *A Generalization of the Divide-sort-merge Strategy for Sorting Networks*, Technical Report No. 16, Digital Systems Laboratory, Stanford University, Stanford, California, August 1971.
8. Van Voorhis, D. C., *Large [g,d] Sorting Networks*, Technical Report

No. 18, Digital Systems Laboratory, Stanford University, Stanford, California, August 1971.

9. Floyd, R. W. and D. E. Knuth, *The Bose-Nelson Sorting Problem*, CS Report 70-177, Stanford University, Stanford, California, November 1970.
10. Knuth, D. E., "Sorting and Searching," *The Art of Computer Programming*, Vol. 3, Addison-Wesley, 1973.
11. Green, M. W., "Some Improvements in Non-adaptive Sorting Algorithms," *Proc. of the Sixth Annual Princeton Conference on Information Sciences and Systems*, 1972, pp. 387-391.
12. Gale, D. and R. K. Karp, "A Phenomenon in the Theory of Sorting," *IEEE Conference Record of the Eleventh Annual Symposium on Switching and Automata Theory*, 1970, pp. 51-59.
13. Van Voorhis, D. C., *An Economical Construction for Sorting Networks*, Working Paper 16/A45 No. 1, IBM System Development Division, Los Gatos, California, January 1974.
14. Van Voorhis, D. C., *A Lower Bound for Sorting Networks that Use the Divide-sort-merge Strategy*, Technical Report No. 17, Digital Systems Laboratory, Stanford University, Stanford, California, August 1971.

Business and industry in the 70's find computer-aided instruction a practical answer to training problems

by EUGENE G. KERR

Western Institute for Science and Technology
Waco, Texas

INTRODUCTION

When one reads the technical literature of an industry or business or attends their national meetings, he is faced with subjects such as "Real Time Systems," "Data Base Systems," "Management Information Systems" and "Teleprocessing Networks." Clearly the focus of many industries and business concerns is the application of "on-line" computing technology to their varied operational problems. A recent survey of business and industrial computer users indicates that the majority of medium and large scale computer users have on-line or teleprocessing systems scheduled in their short range plans.

These new on-line systems have brought new problems or have compounded old problems during their implementation. Historically, user training on new computer based systems has been a substantial problem which often resulted in a shaky start-up of these new applications. The reasons for the problems have been well documented and range from user hostility to inept systems design. These problems have often been compounded by inadequate training of the end users by the computer oriented systems and programming staff.

The effects of inadequately trained user personnel were most often compensated by the availability of systems and programming personnel augmenting and hand-holding the user personnel during the start-up phase. The cost of doing this has increased significantly in the last few years.

On-line teleprocessing systems, in addition, suffer even more severe problems.

- The user is often hundreds of miles from the computer.
- Errors made by the user on the terminal system can require enough additional processing to correct, that this can severely degrade both the external system (Customer Service) and the computer system itself.
- Most of these new teleprocessing systems bring the computer face to face with the user's customer. Under batch systems the customer was shielded by the user from poor training and/or system. But under these new on-line systems, the sins of omission and commission in training and systems design will be full public view.
- The new on-line system very often requires direct knowledge by significantly larger numbers of people. For example, the Bank of Montreal in Canada has a requirement to train over 20,000 people in some phase of operating the terminal. This poses a significantly increased burden on training.
- With the above large numbers of people, computer staff became even less desirable as the training staff. In addition, few organizations have adequate training staff to meet these needs.

Due to the above problems industry and business have searched for a viable alternative. This search led several major businesses to an unexpected conclusion. Computer-Aided Instruction (CAI) provided both an operationally feasible and cost effective way of solving this new training dilemma. Thus, CAI which has existed experimentally in education for over 10 years without ever becomming a cost effective widely used educational tool, has found a practical role in the solution of industrial training problems.

HISTORY

The first large scale application of CAI to Industrial training came within the IBM Corporation in the mid-sixties. This was the training and upgrading of their field engineers first on the IBM 1440 system with Coursewriter I and then on a Model 360/65 running DOS. It is interesting to note that IBM did not offer Coursewriter (a CAI language) until several years later as an independent processor on the 360 series. Also internal IBM staff were pushing the dedicated CAI system (IBM 1500) using Coursewriter. This system had little application to training in business and industry and eventually proved to be too costly for public education and has been phased out.

In the latter part of the sixties most experimentation was still going on in the "Education Field" with very little being done by business and industry. A few of the earliest major teleprocessing users, notably airlines and finance companies seeing the need for some teaching devices on their systems added simple programmed instructions to their operating systems. These were not sophisticated, but did show both a need and gave some experience with CAI type approaches.

A major contributing factor to the lack of use of CAI first by "Education" and subsequently by business and industry was the approaches taken by developers of CAI languages and tools. IBM with Coursewriter set the general mode of CAI language development. By the late 1960's CAI languages had proliferated as many universities and federally funded projects had developed their own versions. The major languages in the early seventies are:

TUTOR—(Project Plato University of Illinois)
PLANIT—(System Development Corporation)
COURSEWRITER III—(IBM)

In addition many users felt that time sharing or interactive languages such as FORTRAN, BASIC or APL were all that a person needed to do CAI and so much course material was developed in these languages. Though these approaches were adequate for research and experimentation in "Education" they were too expensive for widescale use. Even though research demonstrated that many applications of CAI (ranging from simple drill and practice to sophisticated simulations) were educationally effective, their widespread use was hindered by the high cost and operational implementation problems. Thus, even though much of CAI had proven educationally sound, interest in it as a tool and hope for its eventual widespread use was waning in the early seventies. In fact, many educators including industrial trainers had felt that CAI was just another passing fad. Most "Education" researchers had moved on to "CMI" (Computer Managed Instruction) which they felt had wider and more immediate requirement. Most industrial trainers returned to other approaches: PI, Video Tape, Cassettes, Seminars, etc.

NEW DEVELOPMENTS

As discussed earlier, the seventies has led to a significant increase in on-line teleprocessing system in almost all areas of business and industries. In order to meet the needs for adequate training, companies have intensified their standard training approaches. This has proved to be very expensive since usually a large number of specialized trainers are required and much traveling is involved. Another commonly proposed approach has been to provide some learning support as part of the computer system. This has usually involved programming training aids directly into the actual application programs (United Airlines, Dial Finance). Two major problems have arisen with this latter approach.

- This increases the time required to get application programs running and often significantly adds to their complexity.
- Computer programmers are *not* educators and have little aptitude for training requirements. This means that training aids programmed in are often neither adequate nor meaningful.

Research and development people looking at the above problems and the projected increase in on-line systems, set out the following guidelines as a workable solution for a CAI system that would meet the needs of business and industry.

- The normal operational terminal should be able to be used for training with the proviso that such training would not interfere with the normal operation of the on-going production processing on-line environment.
- The computer software for such a system should use a minimum amount of computer resources (10 to 25 thousand characters of memory) and be usable under a wide variety of business and industrial on-line teleprocessing systems (CICS, TSO, IMS, FASTER, APL, etc.) and on a variety of computers.
- The creation of learning material should be able to be done by regular training staff and entered without requirement for computing programmers or computing knowledge.
- The system should provide comprehensive recordkeeping and management information capability which provides the training staff with the control critical to ensuring training success.
- The training portion of the system should provide capability to produce training materials from simple drill and practice to sophisticated simulation all with automatic recordkeeping.
- The training system should allow the easy use of other media and training materials (PI Texts, Films, Video Tapes, Cassettes, etc.) and provide for the recordkeeping and management of these materials.

A number of new software and application techniques have allowed this type of system to be developed and to be interfaced with business and industrial on-line systems. To date a few specialized and experimental systems approach the above design requirements. One of these is a commercially available system called TIME (Terminal Instructional Managed Education). This system was developed from an experimental university CAI system and has the following characteristics.

- Programmed in IBM Assembly Language.
- Structure is based on Fourth Generation Data Base and Information Retrieval Concepts. (This contrasts to the compiler language approach taken by most CAI developers.)
- Totally compatible with business and industrial transaction oriented systems.
- Economical of Core. (Only 10-15,000 characters required.)
- Highly modular. Consists of over 200 small (500 characters) re-entrant processing modules.
- Able to run under numbers terminal control systems (CICS, IMS, FASTER, APL, etc.).
- Runs simultaneously with most application programs.
- Uses any current data bases and/or on-line files such as personnel records, branch record, etc.
- Easy to update and maintain.

Equally important to the use of any system such as

TIME is its usability by non-technical training staff to produce, test and maintain specific training materials for the on-line system. The following characteristics illustrate the major training support features required.

- No computing knowledge required to produce training material.
- Editing and error correction aids are available to the coursewriter.
- Provision is made for the trainer to develop course material ranging from simple drill to complex simulation material.
- Complete records are maintained on all student activity and use of course material.
- Provision for a management mode that allows the trainer not only to manage his instructional environment but gives quantitative information on the readiness of a trainee to use the system.
- Provision for a management mode that allows the trainer to manage other types of instructional materials such as PI, Video Tapes, Films, Cassetts, etc.

CURRENT USES

Since the advent of a commercially available CAI package for business and industry, a wide variety of uses have been made with it. The majority of users concentrated on training terminal operators as their major task. But even in this short time other creative uses of it have been made. The following is a list of industries where CAI is being used and the general types of application.

- Banking
  - Terminal Training
  - Data Entry
  - Management Training
  - Supervisory Training
  - Personnel Policies
  - Introduction to Banking Services
- Life Insurance
  - Data Entry
  - Terminal Training
  - Sales Training
  - Product Introduction
- Manufacturing
  - Warehousing and Inventory
  - Terminal Training
  - Management Training
  - Computer Techniques
  - Creativity (Aero Space Company)
- Food Processing
  - Terminal Training
  - Data Entry
  - Warehousing and Inventory
- Retail
  - Buyer Training
  - Management Training
  - Sales Training
  - Basic Retail Skills Training

In Branch	Centralized	TIME
2 Weeks elapsed	1 Week elapsed	4 Weeks elapsed
40 Hours per trainee	40 Hours per trainee	20 Hours per trainee
Overtime required significant	Use of replacement personnel required	No dislocation or overtime
Impact on branch operation	Maximum branch dislocation	Little impact on branch operation
Rigid problems with illness, turnover, etc.	Rigid problems with illness, turnover, etc.	Very flexible scheduling allowed
2nd Most efficient learning	Least efficient learning	Most efficient learning
Evaluation subjective	Evaluation subjective	Definite evaluation
Readiness subjective	Readiness subjective	Quantitative information on readiness to convert
Motivation good	Motivation dependent on trainer	High motivation
Management information depends on observation	Little management information available	Management information available on all aspects of training

Figure 1—Operational comparisons

As wider use of CAI is made in business and industry the diversity of materials will expand.

In order to get a better prospective of the use of a CAI system in a business and industrial environment let's look at a summary of an early leader in this field.

Bank of Montreal—Montreal, Quebec

Bank of Montreal

Type of Organization	Nationwide Bank
Number of Locations	Over 1,000
Number of Employees to be Trained	20-30,000 initially
Average Branch	10 staff 6 tellers 1 supervisory 3 managers
Computer Hardware	IBM 370/168's
Communication	Leased Line Network
Computer Software	
Terminal Control System	Developed Internally
Applications	DDA, Savings, Installment Credit
Training System	TIME
Number of Courses	Over 100

This bank was faced with a large problem. They had over 20,000 employees to train at over 1,000 sites. Since they were putting 5,000 terminals on-line, CAI offered an economically and operationally feasible approach. Robert McDougal, Vice-President, responsible for the mechanization project decided that the cost savings (up to $1,000 in some branches) and the operational benefits (Figure 1) more than offset the newness of the concept. His foresight has proven valid in retrospect as the Bank of Montreal is now heavily into conversion.

Being first posed some problems, but the bank drew in its own training staff and hired Gordon Davies with a heavy background in training systems in the military to head the development of the system. Courses were developed and tested by this staff and then were piloted with the first few branches. Since CAI allows changes easily, the bank is still changing materials when they are found not to be doing the job.

After their entry application of CAI the bank pointed out the following benefits beyond the obvious ones of cost and operational use.

- Availability of CAI for other types of training (Management Procedures, Sales, etc.).
- Uniformity of materials and methodology in the presentation of learning material.
- Validation of training materials with easy modification when required.

When the Bank of Montreal completes their training they will be one of the largest users of CAI in any environment business, military or educational.

CONCLUSIONS

CAI has indeed found a permanent place in business and industry. It is predicted that most companies going to major on-line systems will make use of this tool. As users become more experienced they will branch out into wider areas of training. Perhaps this wider application of CAI will spur educational institutions to reevaluate their positions on its use. In addition, having CAI tools available in many major companies opens new vistas for educators to serve the business and industrial areas.

The role of computer assisted instruction (CAI) in management information systems

by RAYMOND J. COLLINS

Kraftco Corporation
Glenview, Illinois

In recent months there has been much discussion of the "Fourth Generation" in computer technology. Defined as a *total system*, not a new series of computer hardware, the Fourth Generation has the following characteristics:

- An interactive communications system
- A data base system
- A transaction processing system

At Kraftco Corporation a new dimension, Computer Assisted Instruction (CAI), has been added to this Fourth Generation total system concept—a dimension which provides management with a new body of information on which to base its decisions. The objective of this paper is to describe how Kraftco, faced with the challenge of implementing a Fourth Generation Order Entry System, utilized a sophisticated CAI system in the management decision-making process.

BACKGROUND

In 1970, the Corporation approved a recommendation to implement a Fourth Generation Order Entry System (OES). This new system would replace the existing paper tape system and provide management with a comprehensive data base with which to plan, evaluate and forecast the directions best suited to the corporate objectives. OES affected every aspect of the business; sales districts, plants and distribution centers, division as well as corporate headquarters, would participate in the system. The specifications called for the utilization of the IBM 3270 Information Display terminals for the entry of order data and IBM 3286 printers for the output of hard copy messages.

In addition, locations with high volume printing requirements (such as distribution centers) would utilize high speed IBM 3780 Card Reader/Printer terminals. Kraftco would be the first company in the food industry to utilize these cathode ray tube (CRT) terminals in an on-line order entry system environment—an environment which included users spread across the United States from Maine to Hawaii and from Alaska to Miami, in Canada from Vancouver to Montreal, and even in Puerto Rico. There would be approximately 140 CRT terminals installed in over 65 locations.

Truly, this multi-million dollar system represented a major challenge both to the management of the corporate Systems Services who must develop the computer system and to the management of the company divisions who must implement and smoothly integrate this new system into the

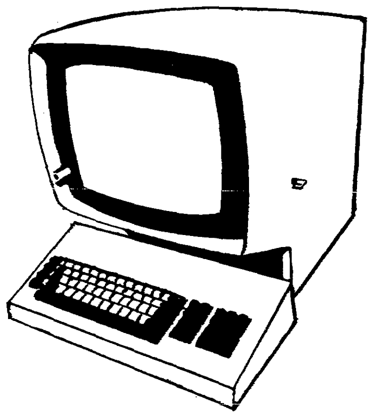


Figure 1—IBM 3270 CRT terminal and keyboard

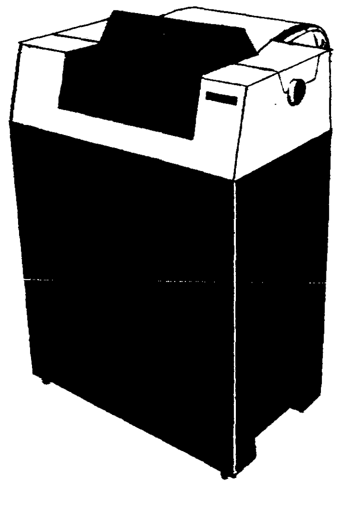


Figure 2—IBM 3286 printer

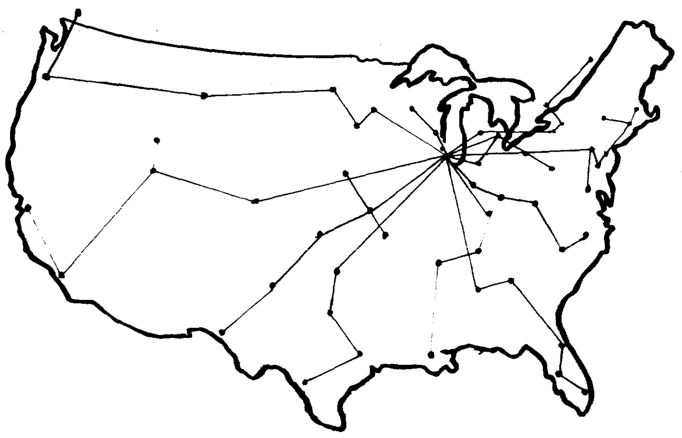


Figure 3—Communications network map

everyday life of the business. Both management teams agreed that prior experience dictated that the system would only be as good as the education and training of its users.

USER TRAINING CONSIDERATIONS

The audience for the education program consisted of two categories of people. First, there were the local management personnel who would be responsible for the implementation of the new Order Entry System at their respective Sales Districts and Distribution Centers. They must understand the new system and be in a position to administer the required training to the administrative personnel who would actually be doing the work. Secondly, there were two groups of administrative personnel:

(1) Sales District, and
(2) Distribution Center.

Each group must be taught how to perform their day-to-day order entry and order dispatching functions utilizing the new CRT terminals and printers. By far, the largest audience was in the second group. Over four hundred administrative personnel must be trained to input the pre-coded transactions on to the appropriately formatted CRT screen and achieve an acceptable level of speed and accuracy while performing this data entry function. An analysis of the problems associated with the training of the OES users came to the following conclusions:

(1) The smooth conversion and implementation of OES are highly dependent on the successful completion of training. Since OES affects many vital areas of the business, all personnel must execute their functions properly.
(2) Errors are especially costly when dealing with real-time systems. Each error compounds file requirements, communication line utilization, error report generation and system control balancing as well as the time of the individuals involved in correcting the error. Thus, a definite economic value could be attached to the benefits realized by a sound, performance-tested education program. The education program had to be designed so that each trainee had to progress beyond *intellectual* understanding of the system to develop his *performance* capabilities.
(3) Centralized training would be extremely difficult because of the diverse geographical distribution of the users. In addition to the cost aspects of travel and living expenses for literally hundreds of users, the interruption of business due to the unavailability of employees who would be attending classes was not desirable.
(4) Since OES would be continually in a state of revision as development and testing progressed simultaneously with user training, students who had been trained early in the cycle must be advised of the latest procedural changes.
(5) If local management were given total responsibility for training, the everyday pressures of running the business could have a negative impact on the quantity and quality of the resulting training program. Local management needed support at a local level to get the job done effectively.
(6) Finally, management must have facts to make the following decisions: Is a particular location *ready* to convert to the new system? Is each location aware of the latest revisions to OES? Will new employees be provided with adequate training *after* a location has been converted to the new system?

To make these decisions, management had to focus its attention on the most basic question of all: What kind of user training should be employed to meet the above challenges?

After a thorough investigation of the most recent developments in industrial training programs, management chose Computer Assisted Instruction as the principal training tool, supplemented in hard copy by comprehensive procedure manuals. The selection of CAI enabled the corporation to insist on a central prerequisite for the education program: that the Order Entry System training effort run in the same environment as OES daily processing. Although the implications of such a decision are far-reaching, at least two challenges were obvious from the outset: (1) In order to utilize the same environment for training and OES daily operations, CAI would have to run successfully on the new IBM 3270 Cathode Ray Tube terminals which had been chosen for the Order Entry System, and (2) CAI courses would have to simulate the graphic control features which are used so extensively in the Order Entry System formatting techniques.

SYSTEM ENVIRONMENT

Adapting CAI software for use with the IBM 3270 terminals was a challenge welcomed by management; many of the outstanding features of the new Cathode Ray Tube would, management reasoned, enhance the training program as much as they enhance the Order Entry System itself. The large screen of the CRT provided 1920 character positions, sufficient to accommodate a full behavioral unit of instruction. Furthermore, the highlighting feature of the IBM 3270 terminals was as potentially useful for CAI course writing as it was for OES transactions. Several of the advanced features of the CRT terminal (extra program function keys, data entry keyboard design, free movement of the cursor) promised to add significant sophisticated dimensions to traditional CAI course development. The graphic control capabilities of the terminal itself could be adapted for use in CAI, just as they had been incorporated into the Order Entry System formatting techniques.

Following a rather extensive exploration of the existing CAI software systems, a CAI program was selected that seemed adaptable to the IBM 3270 terminals under CICS. Essentially, the program consists of the TIME (Terminal Instruction Managed Education) software package developed by McDonnell Douglas Automation Corporation, with major enhancements provided by Kraftco personnel. The features of the CAI system evolved, through improvements and software adaptations, into a program of attractive and varied capabilities. Among the most advantageous aspects of the new CAI software were:

(1) Frugal use of 370 core (under 15K)
(2) Easy portability
(3) Convenient author language and simplified coding
(4) Ready adaptability to the 3270 CRT terminals under CICS
(5) Incorporation of the student's name in the text, response, and question lines of each course frame
(6) A "Mailbox" feature which permits on-line communication between course authors and individual trainees, as well as all-point bulletins from course authors to all students
(7) A "Calculator" capability allowing the trainee to interrupt his course at any time and tap the computer's mathematical resources to perform a wide variety of complicated calculations

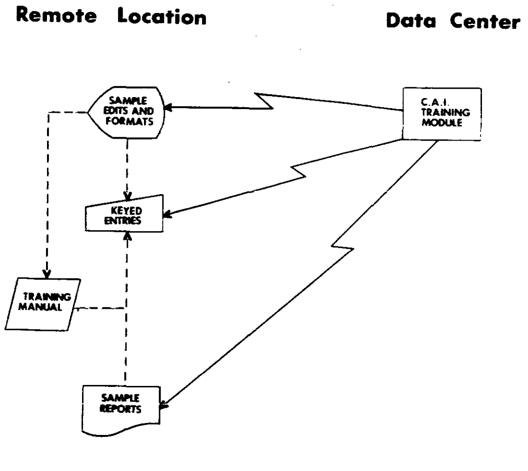
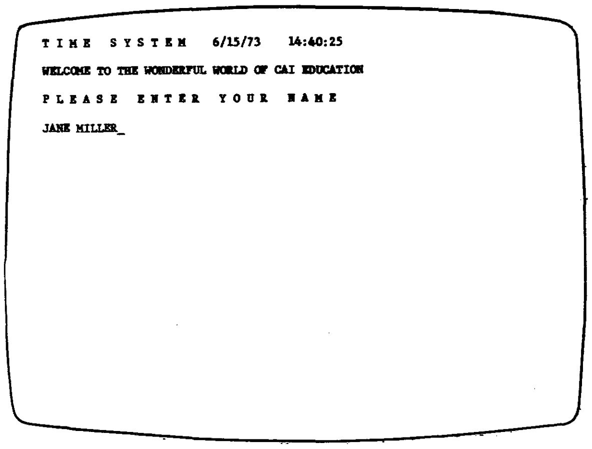


Figure 4—Remote message switching



```
T I M E   S Y S T E M    6/15/73    14:40:25
WELCOME TO THE WONDERFUL WORLD OF CAI EDUCATION
P L E A S E   E N T E R   Y O U R   N A M E
JANE MILLER_
```

Figure 5—Signing into CAI

(8) An "Index" feature which can serve as an on-line dictionary, cross-indexing terms and concepts for quick reference by trainees at any time during a course
(9) A Computer Managed Instruction System (CMI) with sound information processing, readily adaptable to the corporation's own needs.

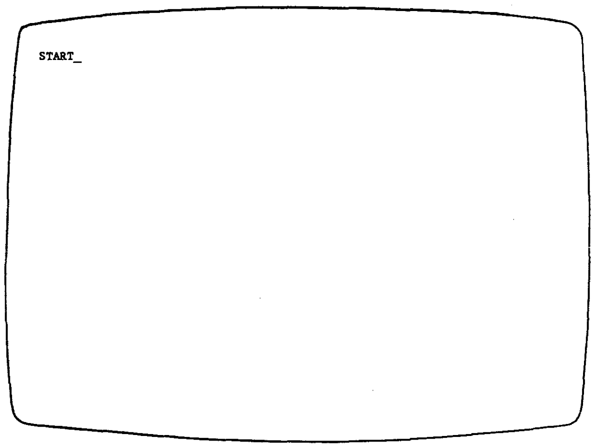
ORGANIZATION OF TRAINING PERSONNEL

Once the initial software for the Computer Assisted Instruction had been acquired, a specialized CAI training team was formulated, consisting of one person with a solid technical background and another with considerable teaching experience in the humanities, but without technical background. The training team composition, while somewhat unusual, was very instrumental in the effectiveness of the CAI effort. Maintaining a staff technical expert on the CAI software development provided management with control of the CAI design modifications—both in liaison with, and independent of, the efforts of McDonnell Douglas Automation Corporation. Entrusting the CAI curriculum development to a trained non-technical educator resulted in courses with a readable "humanized" writing style and a sound educational psychology. In a sincere attempt to avoid duplicating some other companies' mistakes, where programmers as course authors produced CAI courses that were logically sound but tediously boring, Kraftco courses were designed under the supervision of educators and were written to appeal to user trainees who possessed no technical background.

INITIAL PREPARATIONS

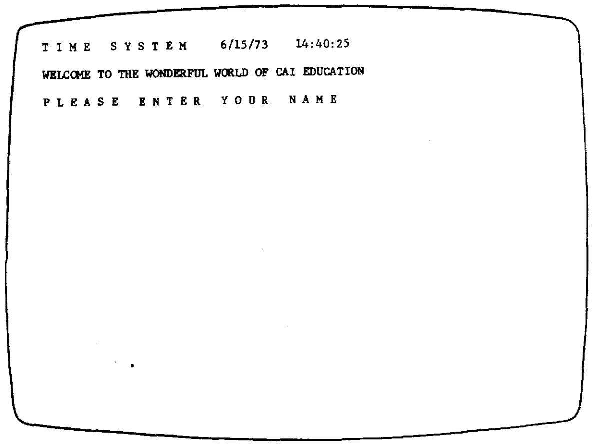
In order to provide a valuable tool for effective course writing, a "CAI Standards Manual" was created for use by the course authors. This manual sets high standards for learning theory, grammatical style, and educational structure

HOW TO BEGIN TAKING A C.A.I. COURSE



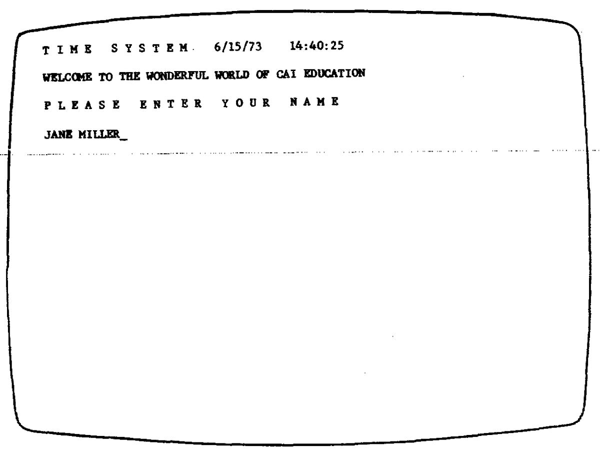
```
START_
```

Figure 6(a)



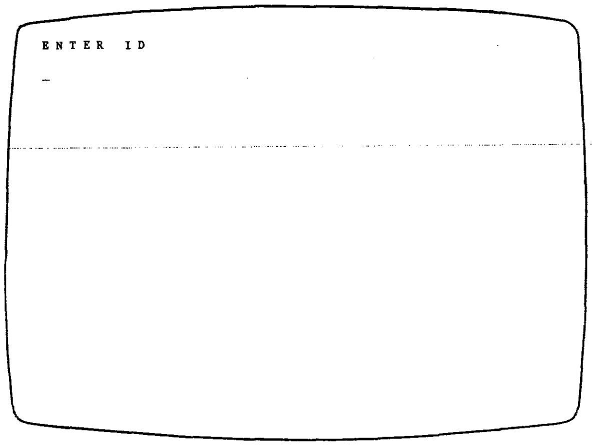
```
T I M E   S Y S T E M    6/15/73    14:40:25
WELCOME TO THE WONDERFUL WORLD OF CAI EDUCATION
P L E A S E   E N T E R   Y O U R   N A M E
```

Figure 6(b)



```
T I M E   S Y S T E M    6/15/73    14:40:25
WELCOME TO THE WONDERFUL WORLD OF CAI EDUCATION
P L E A S E   E N T E R   Y O U R   N A M E
JANE MILLER_
```

Figure 6(c)



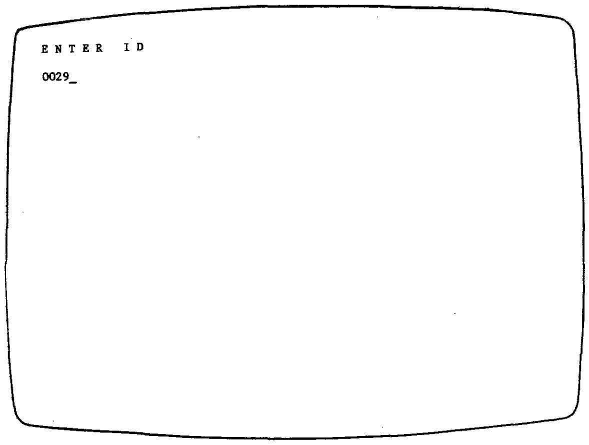
```
E N T E R   I D
_
```

Figure 6(d)

of each course. Emphasis is placed on the importance of sound behavioral objectives as the foundation for each course; in actual practice at Kraftco, behavioral objectives composed by course authors must be approved by the CAI curriculum coordinator before actual course writing may begin. Much of the "CAI Standards Manual" is devoted to exercises and instructions on how to incorporate the modern concepts of totally individualized instruction in CAI course writing: courses are structured according to decision points and branching techniques, permitting experienced and bright students to proceed swiftly through the lesson while trainees requiring more detailed information are also accommodated. "Leap frog learning," or the ability of a student to skip lesson units that do not pertain to him, is one of the outstanding benefits of CAI branching technique, when it is utilized skillfully.

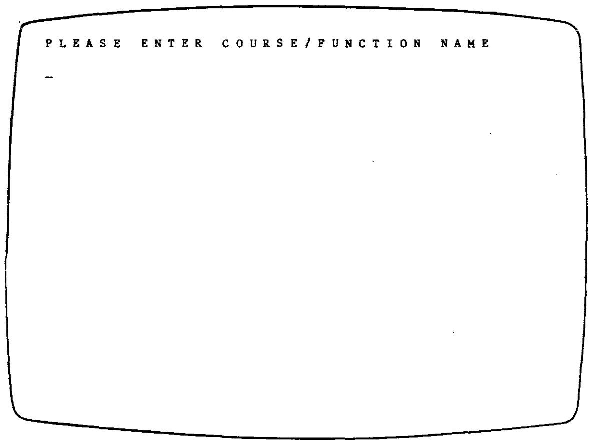
To aid course authors in the creation of interesting and valid instructional units, the "CAI Standards Manual" addresses such topics as: advantageous use of the trainee's name, effective incorporation of student interaction, appropriate audience analysis, the need for a defined "computer teacher" personality, the necessity for clearly stated achievement expectations, the ability to reword the same concept again as an effective remedial measure, the advantages of positive reinforcement and the impact of good negative reinforcement (complete with sample responses, arranged by degree of intensity), and an explanation of the various answer types. While the existence of such a guide to CAI course writing may not be unusual with other companies working in the same field, Kraftco's strong insistence on high educational standards as evidenced by the *comprehensive* nature of the "CAI Standards Manual" is one of the factors surely contributing to the corporation's outstanding success with Computer Assisted Instruction. Furthermore, the "CAI Standards Manual" becomes especially helpful when course authors are programmers or system designers who have expertise in one phase of the Order Entry System, but who lack teaching and writing experience. Although most course writing is done by the full-time CAI training staff, management is able to enlist expert programmers' efforts as temporary course authors by providing them with the manual outlining sound pedagogical practice.

While the educational thrust of the CAI course development was well under way, the technical advancements of the CAI software were increasing rapidly. The technical expert on the training staff worked in close conjunction with the McDonnell Douglas designers to create a CAI program with the most outstanding capabilities possible. Of particular



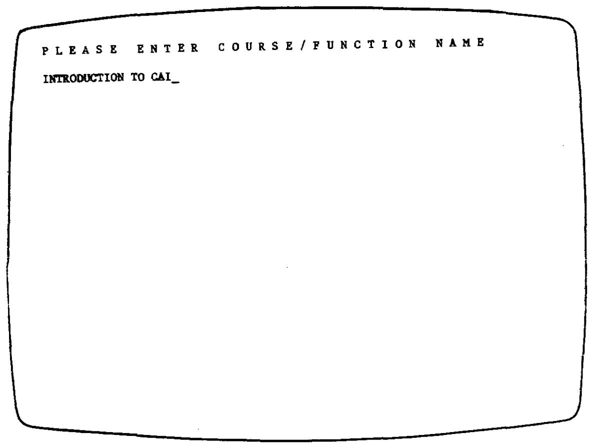
```
E N T E R   I D

0029_
```

Figure 6(e)



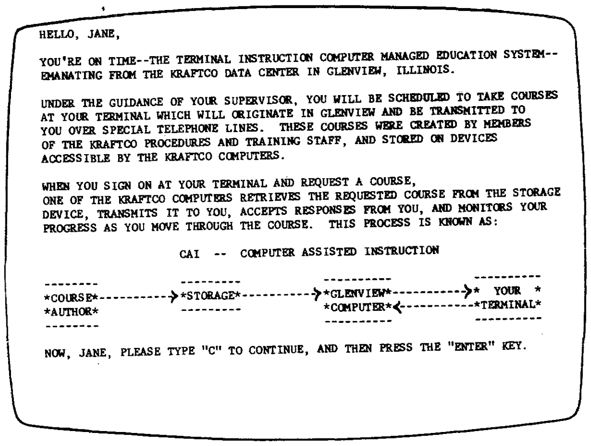
```
P L E A S E   E N T E R   C O U R S E / F U N C T I O N   N A M E

_
```

Figure 6(f)



```
P L E A S E   E N T E R   C O U R S E / F U N C T I O N   N A M E

INTRODUCTION TO CAI_
```

Figure 6(g)



```
HELLO, JANE,

YOU'RE ON TIME--THE TERMINAL INSTRUCTION COMPUTER MANAGED EDUCATION SYSTEM--
EMANATING FROM THE KRAFTCO DATA CENTER IN GLENVIEW, ILLINOIS.

UNDER THE GUIDANCE OF YOUR SUPERVISOR, YOU WILL BE SCHEDULED TO TAKE COURSES
AT YOUR TERMINAL WHICH WILL ORIGINATE IN GLENVIEW AND BE TRANSMITTED TO
YOU OVER SPECIAL TELEPHONE LINES.  THESE COURSES WERE CREATED BY MEMBERS
OF THE KRAFTCO PROCEDURES AND TRAINING STAFF, AND STORED ON DEVICES
ACCESSIBLE BY THE KRAFTCO COMPUTERS.

WHEN YOU SIGN ON AT YOUR TERMINAL AND REQUEST A COURSE,
ONE OF THE KRAFTCO COMPUTERS RETRIEVES THE REQUESTED COURSE FROM THE STORAGE
DEVICE, TRANSMITS IT TO YOU, ACCEPTS RESPONSES FROM YOU, AND MONITORS YOUR
PROGRESS AS YOU MOVE THROUGH THE COURSE.  THIS PROCESS IS KNOWN AS:

                  CAI  --  COMPUTER ASSISTED INSTRUCTION

--------            ---------            ----------            ----------
*COURSE*---------->*STORAGE*---------->*GLENVIEW*---------->*  YOUR  *
*AUTHOR*            ---------            *COMPUTER*<----------*TERMINAL*
--------                                 ----------            ----------

NOW, JANE, PLEASE TYPE "C" TO CONTINUE, AND THEN PRESS THE "ENTER" KEY.
```

Figure 6(h)

Figure 6—How to begin taking a CAI course
(a) After entering a security code for the Order Entry System, the student clears the CRT screen; she then types "START" and hits the "Enter" key on her keyboard; (b) The system responds with heading information for the CAI program, and asks for the student's name; (c) The student types her name, beginning where the cursor is pointing, and then depresses "Enter"; (d) The system requests the student's official identification number; (e) The student records her ID on the screen and presses "Enter"; (f) The system now asks the student to name the course she wishes to take; (g) The student chooses a course from the "Course Catalog" in her CAI Workbook, and types the official course title on the screen. She then presses "Enter"; (h) The first frame of the selected course appears on the screen. If the student had begun this course earlier, the system would have placed her in the last frame she had completed, rather than at the beginning of the course

value was the addition of the following programming features to the existing CAI software:

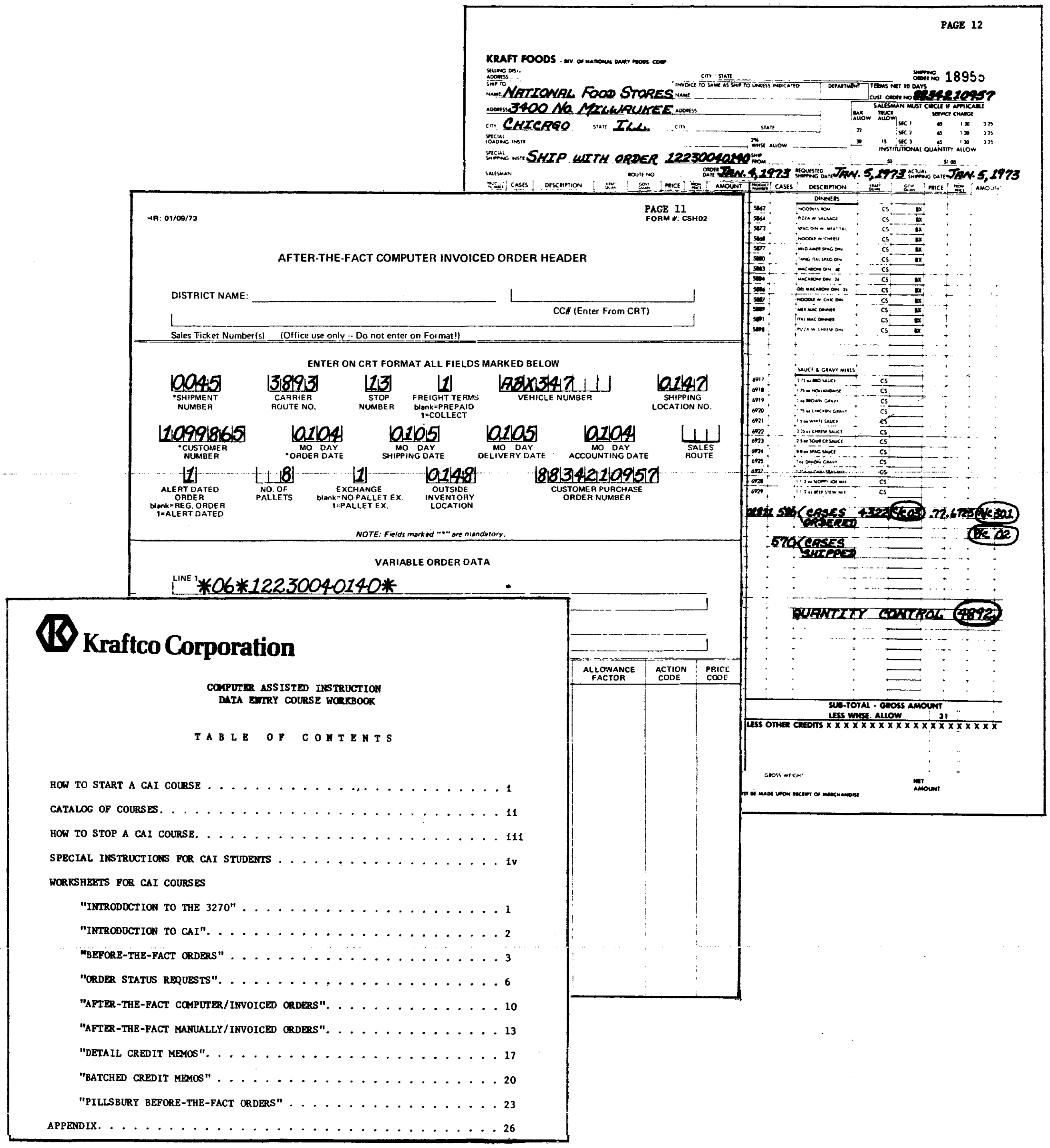
(1) Macro-type expansion pre-compiler utility
(2) Standardized course flow control
(3) Diagnostic editor
(4) Course flowcharter which diagnoses logic flow errors and provides a course flowchart for documentation purposes
(5) Report generator module
(6) Copy library capability for course authors, utilizing Panvalet

COURSE DEVELOPMENT PROCEDURE

Once the technical and educational preparations for CAI course development were made, a logical procedure for all course construction was devised. Each course author initially prepares a series of behavioral objectives for his course; once the objectives are approved by the CAI curriculum coordinator, the course author composes his course. Each completed course is referred to the designer of the specific application in question for content verification, and is then passed to the curriculum coordinator for a review of the style and psychology. Finally, the CAI technical expert verifies that the course makes use of the latest CAI technical capabilities. Where additional programming needs are recognized, the technical man formulates proposals for further software development by McDonnell Douglas Automation Corporation, or he creates programs himself to meet special requirements. The organized course writing procedure worked smoothly; in less than two months, eight full-length CAI courses were developed.

While seven of the eight courses centered on the major

SAMPLE PAGES FROM C.A.I. WORKBOOK



PAGE 12

KRAFT FOODS - DIV. OF NATIONAL DAIRY PRODS. CORP.

SELLING DIST. ADDRESS CITY / STATE SHIPPING ORDER NO 18955

SHIP TO NAME NATIONAL FOOD STORES INVOICE TO SAME AS SHIP TO UNLESS INDICATED NAME DEPARTMENT TERMS NET 10 DAYS CUST ORDER NO 8834210957

ADDRESS 3400 No. MILWAUKEE ADDRESS

CITY CHICAGO STATE ILL. CITY STATE

SALESMAN MUST CIRCLE IF APPLICABLE

BAK ALLOW	TRUCK ALLOW	SERVICE CHARGE			
		SEC 1	65	1 30	3 25
77		SEC 2	65	1 30	3 25
30	15	SEC 3	65	1 30	3 25

INSTITUTIONAL QUANTITY ALLOW 50 $1 00

SPECIAL LOADING INSTR 2% WHSE ALLOW

SPECIAL SHIPPING INSTR SHIP WITH ORDER 1223004040 SHIP FROM

SALESMAN ROUTE NO ORDER DATE JAN. 4, 1973 REQUESTED SHIPPING DATE JAN. 5, 1973 ACTUAL SHIPPING DATE JAN. 5, 1973

PRODUCT NUMBER	CASES	DESCRIPTION	KRAFT QUAN	GOVT QUAN	PRICE	FROM PRICE	AMOUNT
		DINNERS					
5862		NOODLES ROM	CS	BX			
5864		PIZZA W SAUSAGE	CS	BX			
5873		SPAG DIN W MEAT SAU	CS	BX			
5868		NOODLE W CHEESE	CS	BX			
5877		MILD AMER SPAG DIN	CS	BX			
5880		TANG ITAL SPAG DIN	CS	BX			
5883		MACARONI DIN 48	CS				
5884		MACARONI DIN 24	CS	BX			
5886		DEL MACARONI DIN 24	CS	BX			
5887		NOODLE W CHIC DIN	CS	BX			
5889		MEX MAC DINNER	CS	BX			
5891		ITAL MAC DINNER	CS	BX			
5898		PIZZA W CHEESE DIN	CS	BX			
		SAUCE & GRAVY MIXES					
6917		2.75 oz BBQ SAUCE	CS				
6918		1.75 oz HOLLANDAISE	CS				
6919		1 oz BROWN GRAVY	CS				
6920		.75 oz CHICKEN GRAVY	CS				
6921		1.5 oz WHITE SAUCE	CS				
6922		2.25 oz CHEESE SAUCE	CS				
6923		2.5 oz SOUR CR SAUCE	CS				
6924		8.8 oz SPAG SAUCE	CS				
6925		1 oz ONION GRAVY	CS				
6927		1.75 oz CHILI SEAS MIX	CS				
6928		1.1 2 oz SLOPPY JOE MIX	CS				
6929		1.1 2 oz BEEF STEW MIX	CS				

[illegible] 586 CASES ORDERED 4322 SK03 .77.6725 NC 301

RC 02

570 CASES SHIPPED

QUANTITY CONTROL 4892

SUB-TOTAL - GROSS AMOUNT

LESS WHSE. ALLOW 31

LESS OTHER CREDITS X X X X X X X X X X X X X X X X X X X X

GROSS WEIGHT NET AMOUNT

...ST BE MADE UPON RECEIPT OF MERCHANDISE

HR: 01/09/73

PAGE 11
FORM #: CSH02

AFTER-THE-FACT COMPUTER INVOICED ORDER HEADER

DISTRICT NAME: ______ CC# (Enter From CRT)

Sales Ticket Number(s) (Office use only -- Do not enter on Format!)

ENTER ON CRT FORMAT ALL FIELDS MARKED BELOW

0045 *SHIPMENT NUMBER	3893 CARRIER ROUTE NO.	13 STOP NUMBER	1 FREIGHT TERMS blank=PREPAID 1=COLLECT	ABX347 VEHICLE NUMBER	0147 SHIPPING LOCATION NO.
1099865 *CUSTOMER NUMBER	0104 MO DAY *ORDER DATE	0105 MO DAY SHIPPING DATE	0105 MO DAY DELIVERY DATE	0104 MO DAY ACCOUNTING DATE	SALES ROUTE
1 ALERT DATED ORDER blank=REG. ORDER 1=ALERT DATED	8 NO. OF PALLETS	1 EXCHANGE blank=NO PALLET EX. 1=PALLET EX.	0148 OUTSIDE INVENTORY LOCATION	8834210957 CUSTOMER PURCHASE ORDER NUMBER	

NOTE: Fields marked "" are mandatory.*

VARIABLE ORDER DATA

LINE 1 *06*1223004014O*

ALLOWANCE FACTOR	ACTION CODE	PRICE CODE

Kraftco Corporation

COMPUTER ASSISTED INSTRUCTION
DATA ENTRY COURSE WORKBOOK

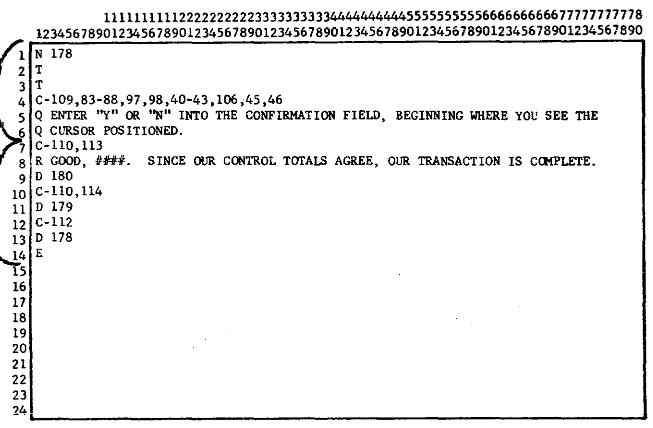
TABLE OF CONTENTS

Figure 7—Sample pages from CAI workbook

AUTHOR CODING

This portion of the frame simulates a typical Order Entry System format. The student is able to enter the answer in the body of the format itself, thereby utilizing CAI graphic controls Panvalet Copy Library statements are used to simplify author coding.

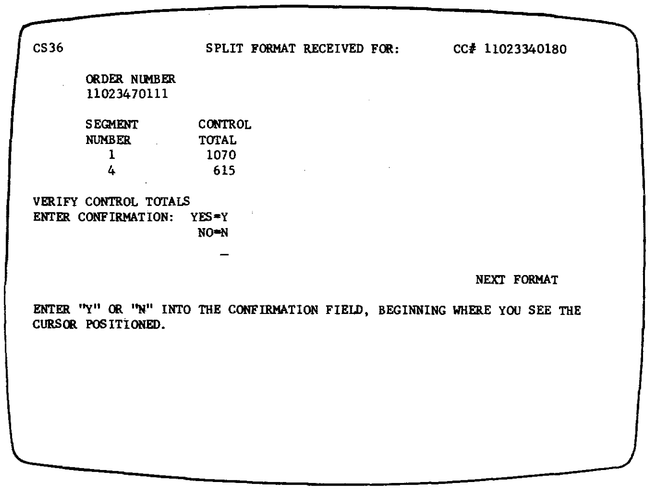
This portion of the frame consists of the various answers, responses, and destinations programmed for this frame. Panvalet Copy Library statements are also utilized here.



```
N 178
T
T
C-109,83-88,97,98,40-43,106,45,46
Q ENTER "Y" OR "N" INTO THE CONFIRMATION FIELD, BEGINNING WHERE YOU SEE THE
Q CURSOR POSITIONED.
C-110,113
R GOOD, ####.  SINCE OUR CONTROL TOTALS AGREE, OUR TRANSACTION IS COMPLETE.
D 180
C-110,114
D 179
C-112
D 178
E
```

RESULTING STUDENT TEXT

The student is able to practice working with an actual Order Entry System format as part of the CAI lesson. The answer is to be entered where the cursor indicates. Depending on the accuracy of the answer, the student will be given an appropriate response and destination, as coded by the author.



```
CS36                    SPLIT FORMAT RECEIVED FOR:        CC# 11023340180

       ORDER NUMBER
       11023470111

       SEGMENT        CONTROL
       NUMBER         TOTAL
          1            1070
          4             615

VERIFY CONTROL TOTALS
ENTER CONFIRMATION:  YES=Y
                      NO=N
                         _

                                                            NEXT FORMAT

ENTER "Y" OR "N" INTO THE CONFIRMATION FIELD, BEGINNING WHERE YOU SEE THE
CURSOR POSITIONED.
```

Figure 8—A sample course frame showing the use of an actual order entry system format

transactions of the Order Entry System, one course served as an introduction to the operation of the IBM 3270 terminal. Initially, the decision was made to treat the subject of *data entry* procedures for each of the seven transactions, since data entry skill was required by hundreds of operators throughout the country. Other topics requiring training—such as how to code orders for the various Order Entry System transactions—pertain to far fewer "students" across the nation; the decision was made, therefore, to teach those subjects by alternate methods, such as week-long classes at headquarters or special sections in the procedure manuals. CAI courses were to focus on the *data entry* techniques only.*

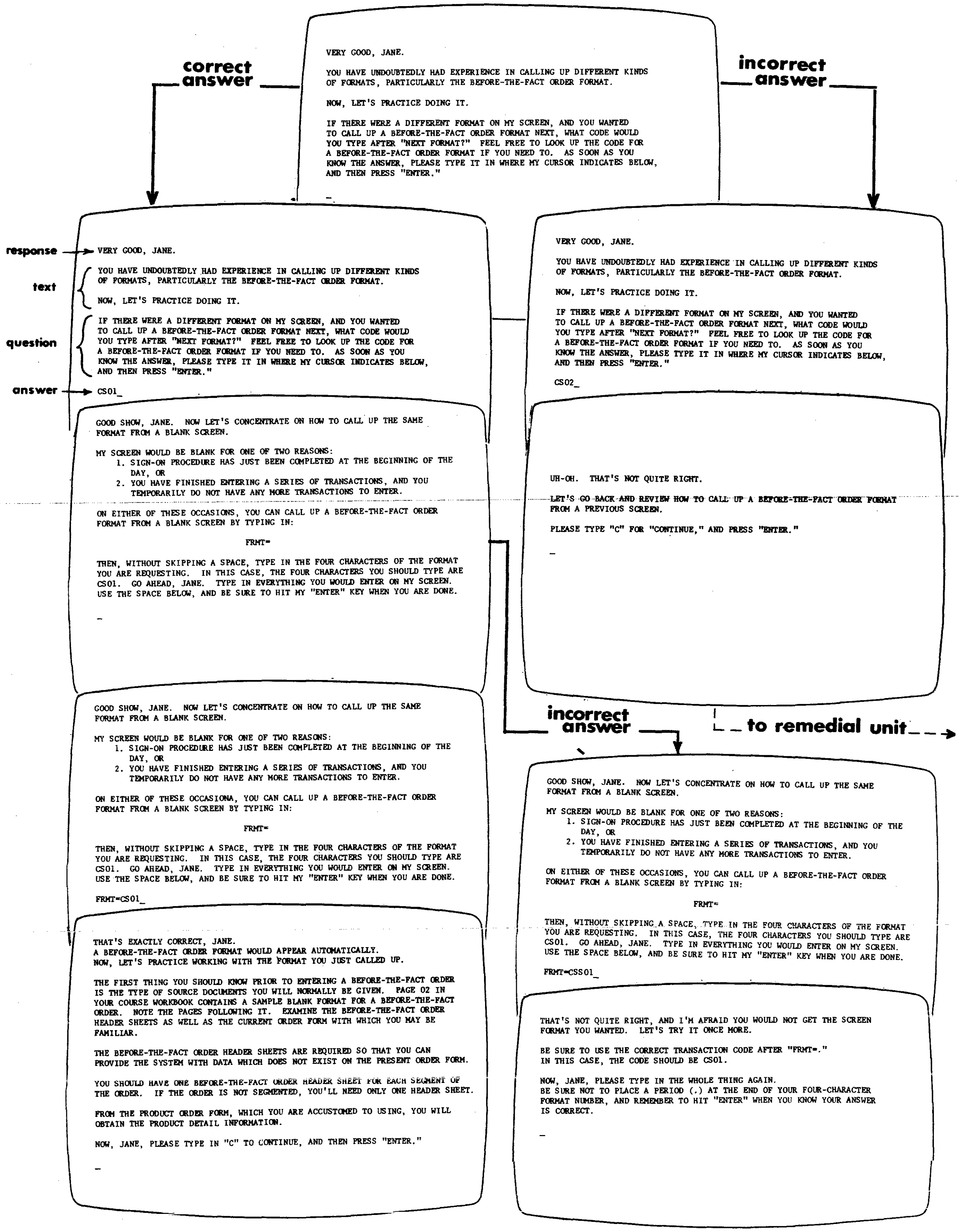
The average length of a CAI course in OES data entry is 100 frames. Each course went through approximately four revisions in the first two months of development, while CAI software was simultaneously adapted to bring all the courses to the IBM 3270 screens. Although there were still technical and contextual revisions to make, the training staff met a very tight deadline successfully.

Now—six months later—each of those original eight courses has been updated at least fourteen times to incorporate the latest system changes, technical capabilities, and educational approaches. In addition, six more courses have been added to the catalog, a dictionary of terms for the CAI Index is being compiled, and the "CAI Standards Manual" is being expanded.

Trainees followed a specified sign-on procedure which records pertinent data for subsequent Computer Managed Instruction (CMI) reporting.

Meanwhile, employees across the nation are enjoying considerable exposure to CAI training. Over two hundred seventy-five trainees have already registered their names and identification numbers with the computer so that they

* The corporation conducted six user classes at a central location to train office managers in the coding of order forms for the Order Entry System. The CAI training team members were fully prepared to serve as class instructors, since their course authoring experience had made them thoroughly familiar with the details of the Order Entry System. During the week-long classes on order coding for OES, the local office managers were given ten hours or more of on-line exposure to the CAI courses on data entry. In this way, the office managers were able to participate briefly in the CAI training program which had been prepared for their local employees.



```
VERY GOOD, JANE.

YOU HAVE UNDOUBTEDLY HAD EXPERIENCE IN CALLING UP DIFFERENT KINDS
OF FORMATS, PARTICULARLY THE BEFORE-THE-FACT ORDER FORMAT.

NOW, LET'S PRACTICE DOING IT.

IF THERE WERE A DIFFERENT FORMAT ON MY SCREEN, AND YOU WANTED
TO CALL UP A BEFORE-THE-FACT ORDER FORMAT NEXT, WHAT CODE WOULD
YOU TYPE AFTER "NEXT FORMAT?"  FEEL FREE TO LOOK UP THE CODE FOR
A BEFORE-THE-FACT ORDER FORMAT IF YOU NEED TO.  AS SOON AS YOU
KNOW THE ANSWER, PLEASE TYPE IT IN WHERE MY CURSOR INDICATES BELOW,
AND THEN PRESS "ENTER."

_
```

```
VERY GOOD, JANE.

YOU HAVE UNDOUBTEDLY HAD EXPERIENCE IN CALLING UP DIFFERENT KINDS
OF FORMATS, PARTICULARLY THE BEFORE-THE-FACT ORDER FORMAT.

NOW, LET'S PRACTICE DOING IT.

IF THERE WERE A DIFFERENT FORMAT ON MY SCREEN, AND YOU WANTED
TO CALL UP A BEFORE-THE-FACT ORDER FORMAT NEXT, WHAT CODE WOULD
YOU TYPE AFTER "NEXT FORMAT?"  FEEL FREE TO LOOK UP THE CODE FOR
A BEFORE-THE-FACT ORDER FORMAT IF YOU NEED TO.  AS SOON AS YOU
KNOW THE ANSWER, PLEASE TYPE IT IN WHERE MY CURSOR INDICATES BELOW,
AND THEN PRESS "ENTER."

CS01_
```

```
VERY GOOD, JANE.

YOU HAVE UNDOUBTEDLY HAD EXPERIENCE IN CALLING UP DIFFERENT KINDS
OF FORMATS, PARTICULARLY THE BEFORE-THE-FACT ORDER FORMAT.

NOW, LET'S PRACTICE DOING IT.

IF THERE WERE A DIFFERENT FORMAT ON MY SCREEN, AND YOU WANTED
TO CALL UP A BEFORE-THE-FACT ORDER FORMAT NEXT, WHAT CODE WOULD
YOU TYPE AFTER "NEXT FORMAT?"  FEEL FREE TO LOOK UP THE CODE FOR
A BEFORE-THE-FACT ORDER FORMAT IF YOU NEED TO.  AS SOON AS YOU
KNOW THE ANSWER, PLEASE TYPE IT IN WHERE MY CURSOR INDICATES BELOW,
AND THEN PRESS "ENTER."

CS02_
```

```
GOOD SHOW, JANE.  NOW LET'S CONCENTRATE ON HOW TO CALL UP THE SAME
FORMAT FROM A BLANK SCREEN.

MY SCREEN WOULD BE BLANK FOR ONE OF TWO REASONS:
    1. SIGN-ON PROCEDURE HAS JUST BEEN COMPLETED AT THE BEGINNING OF THE
       DAY, OR
    2. YOU HAVE FINISHED ENTERING A SERIES OF TRANSACTIONS, AND YOU
       TEMPORARILY DO NOT HAVE ANY MORE TRANSACTIONS TO ENTER.

ON EITHER OF THESE OCCASIONS, YOU CAN CALL UP A BEFORE-THE-FACT ORDER
FORMAT FROM A BLANK SCREEN BY TYPING IN:

                             FRMT=

THEN, WITHOUT SKIPPING A SPACE, TYPE IN THE FOUR CHARACTERS OF THE FORMAT
YOU ARE REQUESTING.  IN THIS CASE, THE FOUR CHARACTERS YOU SHOULD TYPE ARE
CS01.  GO AHEAD, JANE.  TYPE IN EVERYTHING YOU WOULD ENTER ON MY SCREEN.
USE THE SPACE BELOW, AND BE SURE TO HIT MY "ENTER" KEY WHEN YOU ARE DONE.

_
```

```
UH-OH.  THAT'S NOT QUITE RIGHT.

LET'S GO BACK AND REVIEW HOW TO CALL UP A BEFORE-THE-FACT ORDER FORMAT
FROM A PREVIOUS SCREEN.

PLEASE TYPE "C" FOR "CONTINUE," AND PRESS "ENTER."

_
```

```
GOOD SHOW, JANE.  NOW LET'S CONCENTRATE ON HOW TO CALL UP THE SAME
FORMAT FROM A BLANK SCREEN.

MY SCREEN WOULD BE BLANK FOR ONE OF TWO REASONS:
    1. SIGN-ON PROCEDURE HAS JUST BEEN COMPLETED AT THE BEGINNING OF THE
       DAY, OR
    2. YOU HAVE FINISHED ENTERING A SERIES OF TRANSACTIONS, AND YOU
       TEMPORARILY DO NOT HAVE ANY MORE TRANSACTIONS TO ENTER.

ON EITHER OF THESE OCCASIONA, YOU CAN CALL UP A BEFORE-THE-FACT ORDER
FORMAT FROM A BLANK SCREEN BY TYPING IN:

                             FRMT=

THEN, WITHOUT SKIPPING A SPACE, TYPE IN THE FOUR CHARACTERS OF THE FORMAT
YOU ARE REQUESTING.  IN THIS CASE, THE FOUR CHARACTERS YOU SHOULD TYPE ARE
CS01.  GO AHEAD, JANE.  TYPE IN EVERYTHING YOU WOULD ENTER ON MY SCREEN.
USE THE SPACE BELOW, AND BE SURE TO HIT MY "ENTER" KEY WHEN YOU ARE DONE.

FRMT=CS01_
```

```
GOOD SHOW, JANE.  NOW LET'S CONCENTRATE ON HOW TO CALL UP THE SAME
FORMAT FROM A BLANK SCREEN.

MY SCREEN WOULD BE BLANK FOR ONE OF TWO REASONS:
    1. SIGN-ON PROCEDURE HAS JUST BEEN COMPLETED AT THE BEGINNING OF THE
       DAY, OR
    2. YOU HAVE FINISHED ENTERING A SERIES OF TRANSACTIONS, AND YOU
       TEMPORARILY DO NOT HAVE ANY MORE TRANSACTIONS TO ENTER.

ON EITHER OF THESE OCCASIONS, YOU CAN CALL UP A BEFORE-THE-FACT ORDER
FORMAT FROM A BLANK SCREEN BY TYPING IN:

                             FRMT=

THEN, WITHOUT SKIPPING A SPACE, TYPE IN THE FOUR CHARACTERS OF THE FORMAT
YOU ARE REQUESTING.  IN THIS CASE, THE FOUR CHARACTERS YOU SHOULD TYPE ARE
CS01.  GO AHEAD, JANE.  TYPE IN EVERYTHING YOU WOULD ENTER ON MY SCREEN.
USE THE SPACE BELOW, AND BE SURE TO HIT MY "ENTER" KEY WHEN YOU ARE DONE.

FRMT=CSS01_
```

```
THAT'S EXACTLY CORRECT, JANE.
A BEFORE-THE-FACT ORDER FORMAT WOULD APPEAR AUTOMATICALLY.
NOW, LET'S PRACTICE WORKING WITH THE FORMAT YOU JUST CALLED UP.

THE FIRST THING YOU SHOULD KNOW PRIOR TO ENTERING A BEFORE-THE-FACT ORDER
IS THE TYPE OF SOURCE DOCUMENTS YOU WILL NORMALLY BE GIVEN.  PAGE 02 IN
YOUR COURSE WORKBOOK CONTAINS A SAMPLE BLANK FORMAT FOR A BEFORE-THE-FACT
ORDER.  NOTE THE PAGES FOLLOWING IT.  EXAMINE THE BEFORE-THE-FACT ORDER
HEADER SHEETS AS WELL AS THE CURRENT ORDER FORM WITH WHICH YOU MAY BE
FAMILIAR.

THE BEFORE-THE-FACT ORDER HEADER SHEETS ARE REQUIRED SO THAT YOU CAN
PROVIDE THE SYSTEM WITH DATA WHICH DOES NOT EXIST ON THE PRESENT ORDER FORM.

YOU SHOULD HAVE ONE BEFORE-THE-FACT ORDER HEADER SHEET FOR EACH SEGMENT OF
THE ORDER.  IF THE ORDER IS NOT SEGMENTED, YOU'LL NEED ONLY ONE HEADER SHEET.

FROM THE PRODUCT ORDER FORM, WHICH YOU ARE ACCUSTOMED TO USING, YOU WILL
OBTAIN THE PRODUCT DETAIL INFORMATION.

NOW, JANE, PLEASE TYPE IN "C" TO CONTINUE, AND THEN PRESS "ENTER."

_
```

```
THAT'S NOT QUITE RIGHT, AND I'M AFRAID YOU WOULD NOT GET THE SCREEN
FORMAT YOU WANTED.  LET'S TRY IT ONCE MORE.

BE SURE TO USE THE CORRECT TRANSACTION CODE AFTER "FRMT=."
IN THIS CASE, THE CODE SHOULD BE CS01.

NOW, JANE, PLEASE TYPE IN THE WHOLE THING AGAIN.
BE SURE NOT TO PLACE A PERIOD (.) AT THE END OF YOUR FOUR-CHARACTER
FORMAT NUMBER, AND REMEMBER TO HIT "ENTER" WHEN YOU KNOW YOUR ANSWER
IS CORRECT.

_
```

Figure 9—CAI branching technique

can be recognized as students. Then, when CAI is extended to them daily through a system of remote message switching, trainees in distant locations sign on to the terminal indicating that they wish to begin taking a CAI course.

Graphic control features have been added to the CAI software so that regular Order Entry System formats can be used in CAI courses; students are able to enter data in the appropriate fields, just as they will when their location is converted to the Order Entry System.

The interactive mode of CAI training allows the student to participate freely in the learning process. Each course frame consists of five basic parts: the text material, the question which is posed to the student, the student's answer, the appropriate response by the course, and the destination to the next frame of instruction. Since the response and destination are both dependent on the accuracy of the student's answer, each course is individually structured for the student's capabilities.

The trainee can choose to take a course all the way through, or he can "STOP" at any time. Once he types "STOP," the computer remembers where the student left off, and will always return him to the frame he last finished—regardless of how much time goes by before the student resumes the course.

EVALUATION

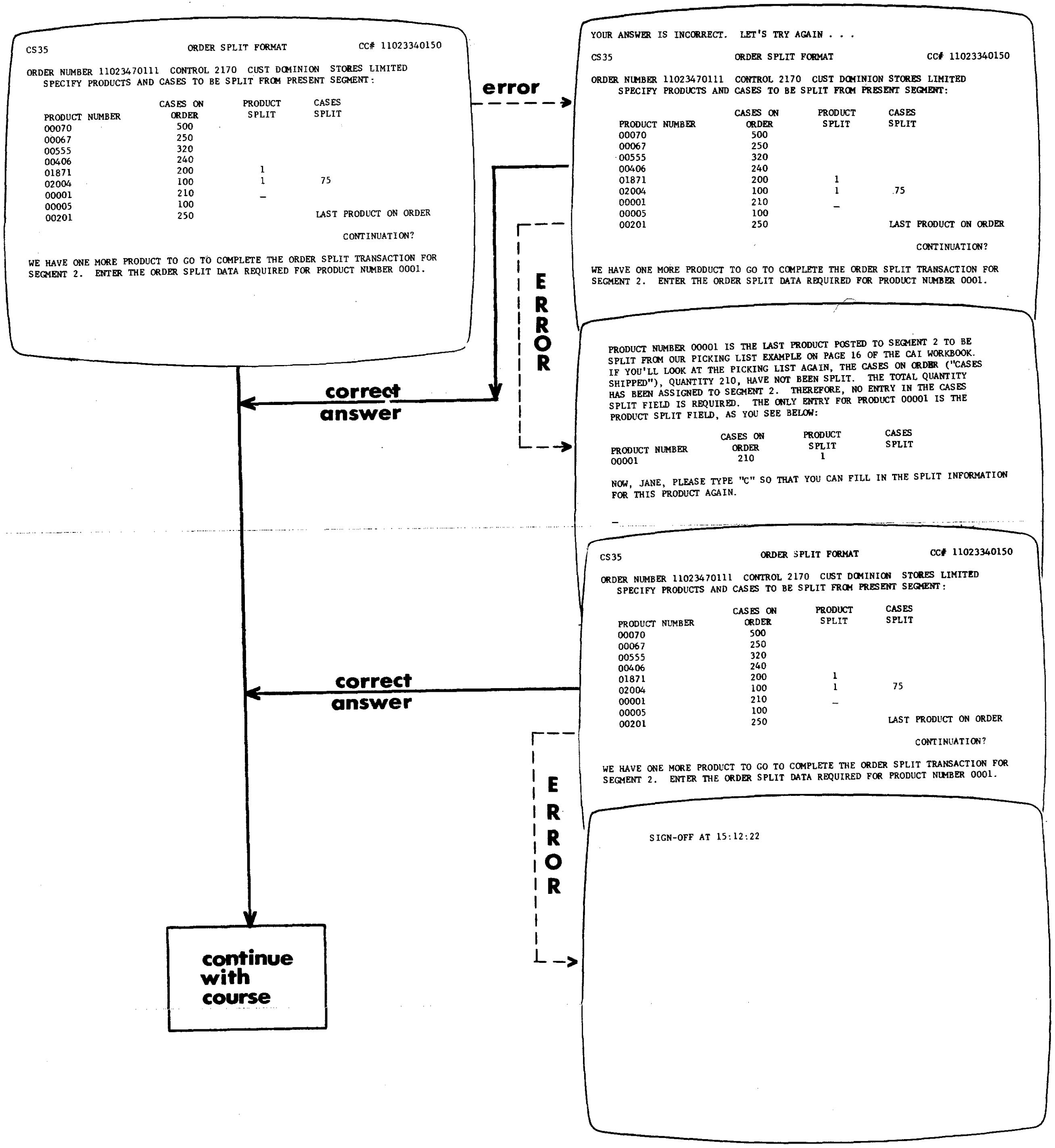
Even at this early stage in the corporation's experience with Computer Assisted Instruction, there has been significant feedback from employees across the country regarding CAI as a major training tool. The most gratifying response has been, unanimously, that learning is enjoyable. Office managers report that their employees look forward to training time each day because the subject itself is made pleasurable and interesting, and the terminal-as-medium is "fascinating." Furthermore, early intrigue with the terminals does not seem to diminish as training pr resses. Another benefit of CAI, from the employees' perspectives, is that the "computer teacher" has been given a distinct and pleasant personality by the CAI authors. The positive attitude of the "teacher" increases the trainees' desires to excel in their learning.

Employees report further that they respond highly favorably to the personalization of the text and question lines of each frame. In addition, the use of the trainee's first name in the positive responses creates an educational atmosphere of interest and trust. On the other hand, empolyees identified a significant error course authors had made. The incorporation of the student's first name in the negative responses seems to be too personal to maintain a healthy learning atmosphere. The psychological impact of "Charlie, I am afraid your answer is wrong again" is too strong, according to the CAI students, because the trainee is usually too embarrassed to try again. Instead, a non-personalized negative response, such as "I am afraid your answer is wrong," permits the student to try again and again without personal guilt over his mistakes. As a result of this feedback from students, the course authors revised all the courses to eliminate names in the negative responses.

One benefit of the CAI training program is that much of the potential apprehension associated with changing from an old system to an entirely new one was removed as a result of the pleasurable exposure to CAI education. From management's perspective, the attitude-shaping capabilities of CAI are a most welcome bonus. Experience to date has shown that employee "fear" of the new equipment is virtually removed by CAI. The early management decision to train students in the same environment used for daily processing—i.e., utilizing the identical equipment in the same physical surroundings—has proved to be a very wise choice. Employees can become familar with the techniques and equipment of the Order Entry System *before* the system is live for their location.

As each office converts from the old system to the new Order Entry System, the most significant benefits of CAI training become obvious. To date, twenty-one locations across the country have been smoothly and efficiently converted to the new Order Entry System without any substantial training problems. The interactive mode of Computer Assisted Instruction has increased student comprehension and concentration to the extent that CAI-trained employees are able to remember and put into effect the precise *details* of OES data entry. Since OES error correction takes place on-line, no statistics on error rates are available. However, the actual number of errors has not been, to date, significant enough to interfere in any way with the smooth conversion to the Order Entry System. CAI has, therefore, proved to be highly successful in teaching discipline and precision by providing practice on sample formats without consequences to the production system, Many of the CAI trainees volunteer that their comprehension seems much greater as a result of taking CAI courses compared to their learning comprehension after reading a procedure manual only. Employees at one location, for example, once noticed a minute change (the nonsignificant addition of a minus sign) that had been made to one frame of a CAI course. They recognized instantly that the frame in question had been updated overnight. When questioned, the trainees readily admitted that they would probably never notice a similar change to the procedure manual they are reading, but that they find it quite easy to concentrate on the small details of CAI courses.

One of the reasons cited for the high degree of comprehension by CAI trainees is that the CAI "computer teacher" is much more helpful in providing detailed remedial information than a traditional teacher can afford to be in the classroom situation—or than a procedure manual can afford to be in its print structure. CAI students can move at their own speed without guilt at proceeding too slowly, and without undue pressure to finish quickly. Whenever additional information is required, it is readily provided before the student can move on to another lesson unit. Unlike one of the dangers of traditional programmed instruction, however, CAI trainees never get "stuck" in one frame if they are having difficulty with an answer. A programmed



```
CS35                        ORDER SPLIT FORMAT              CC# 11023340150

ORDER NUMBER 11023470111  CONTROL 2170  CUST DOMINION  STORES LIMITED
   SPECIFY PRODUCTS AND CASES TO BE SPLIT FROM PRESENT SEGMENT:

                              CASES ON      PRODUCT      CASES
   PRODUCT NUMBER              ORDER         SPLIT       SPLIT
   00070                        500
   00067                        250
   00555                        320
   00406                        240
   01871                        200            1
   02004                        100            1          75
   00001                        210            _
   00005                        100
   00201                        250                     LAST PRODUCT ON ORDER

                                                           CONTINUATION?

WE HAVE ONE MORE PRODUCT TO GO TO COMPLETE THE ORDER SPLIT TRANSACTION FOR
SEGMENT 2.  ENTER THE ORDER SPLIT DATA REQUIRED FOR PRODUCT NUMBER 0001.
```

```
YOUR ANSWER IS INCORRECT.  LET'S TRY AGAIN . . .

CS35                        ORDER SPLIT FORMAT              CC# 11023340150

ORDER NUMBER 11023470111  CONTROL 2170  CUST DOMINION  STORES LIMITED
   SPECIFY PRODUCTS AND CASES TO BE SPLIT FROM PRESENT SEGMENT:

                              CASES ON      PRODUCT      CASES
   PRODUCT NUMBER              ORDER         SPLIT       SPLIT
   00070                        500
   00067                        250
   00555                        320
   00406                        240
   01871                        200            1
   02004                        100            1          75
   00001                        210            _
   00005                        100
   00201                        250                     LAST PRODUCT ON ORDER

                                                           CONTINUATION?

WE HAVE ONE MORE PRODUCT TO GO TO COMPLETE THE ORDER SPLIT TRANSACTION FOR
SEGMENT 2.  ENTER THE ORDER SPLIT DATA REQUIRED FOR PRODUCT NUMBER 0001.
```

```
PRODUCT NUMBER 00001 IS THE LAST PRODUCT POSTED TO SEGMENT 2 TO BE
SPLIT FROM OUR PICKING LIST EXAMPLE ON PAGE 16 OF THE CAI WORKBOOK.
IF YOU'LL LOOK AT THE PICKING LIST AGAIN, THE CASES ON ORDER ("CASES
SHIPPED"), QUANTITY 210, HAVE NOT BEEN SPLIT.  THE TOTAL QUANTITY
HAS BEEN ASSIGNED TO SEGMENT 2.  THEREFORE, NO ENTRY IN THE CASES
SPLIT FIELD IS REQUIRED.  THE ONLY ENTRY FOR PRODUCT 00001 IS THE
PRODUCT SPLIT FIELD, AS YOU SEE BELOW:

                           CASES ON      PRODUCT      CASES
PRODUCT NUMBER              ORDER         SPLIT       SPLIT
00001                        210            1

NOW, JANE, PLEASE TYPE "C" SO THAT YOU CAN FILL IN THE SPLIT INFORMATION
FOR THIS PRODUCT AGAIN.

_
```

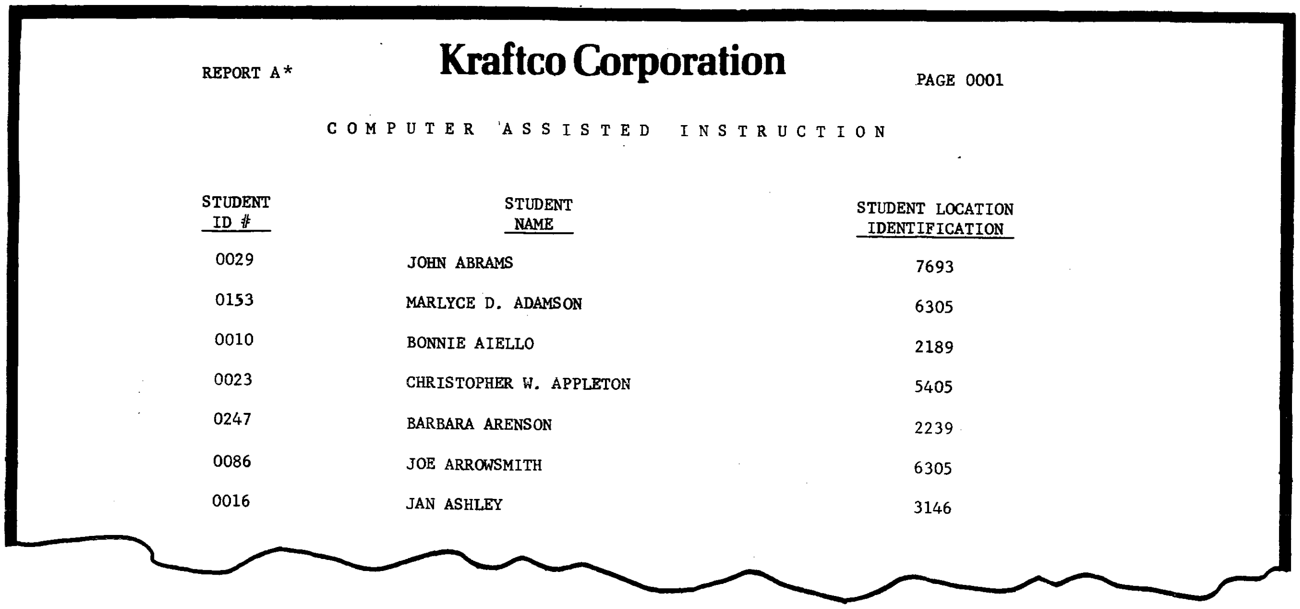
```
CS35                        ORDER SPLIT FORMAT              CC# 11023340150

ORDER NUMBER 11023470111  CONTROL 2170  CUST DOMINION  STORES LIMITED
   SPECIFY PRODUCTS AND CASES TO BE SPLIT FROM PRESENT SEGMENT:

                              CASES ON      PRODUCT      CASES
   PRODUCT NUMBER              ORDER         SPLIT       SPLIT
   00070                        500
   00067                        250
   00555                        320
   00406                        240
   01871                        200            1
   02004                        100            1          75
   00001                        210            _
   00005                        100
   00201                        250                     LAST PRODUCT ON ORDER

                                                           CONTINUATION?

WE HAVE ONE MORE PRODUCT TO GO TO COMPLETE THE ORDER SPLIT TRANSACTION FOR
SEGMENT 2.  ENTER THE ORDER SPLIT DATA REQUIRED FOR PRODUCT NUMBER 0001.
```

```
SIGN-OFF AT 15:12:22
```

Figure 10—Programmed routine for remedial help

routine was designed to accommodate those students who may repeatedly commit the identical error (in spite of three levels of remedial help). The students are always permitted to leave the course without penalty and to seek the answer from the procedure manual or their supervisor. Once the trainee has the answer, he is allowed to re-enter the course at the beginning of the lesson unit he was taking. There is no need for him to start at the beginning of the entire course, because he would then be repeating what he already knows.

As a result of the high degree of success of the Computer

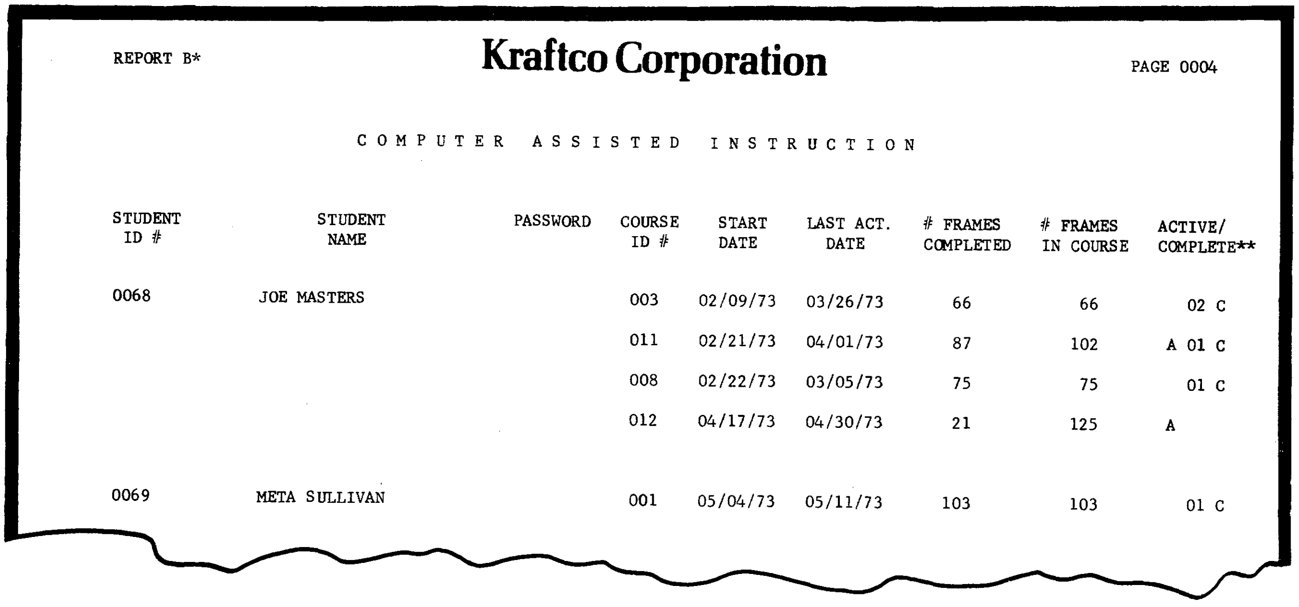


REPORT A* **Kraftco Corporation** PAGE 0001

COMPUTER ASSISTED INSTRUCTION

STUDENT ID #	STUDENT NAME	STUDENT LOCATION IDENTIFICATION
0029	JOHN ABRAMS	7693
0153	MARLYCE D. ADAMSON	6305
0010	BONNIE AIELLO	2189
0023	CHRISTOPHER W. APPLETON	5405
0247	BARBARA ARENSON	2239
0086	JOE ARROWSMITH	6305
0016	JAN ASHLEY	3146

* ALPHABETICAL CLASS LIST



REPORT B* **Kraftco Corporation** PAGE 0004

COMPUTER ASSISTED INSTRUCTION

STUDENT ID #	STUDENT NAME	PASSWORD	COURSE ID #	START DATE	LAST ACT. DATE	# FRAMES COMPLETED	# FRAMES IN COURSE	ACTIVE/ COMPLETE**
0068	JOE MASTERS		003	02/09/73	03/26/73	66	66	02 C
			011	02/21/73	04/01/73	87	102	A 01 C
			008	02/22/73	03/05/73	75	75	01 C
			012	04/17/73	04/30/73	21	125	A
0069	META SULLIVAN		001	05/04/73	05/11/73	103	103	01 C

* COURSE PROGRESS BY STUDENT ID #

** AN "A" ("ACTIVE") CODE INDICATES THAT THE STUDENT IS STILL TAKING THE COURSE.

A "C" ("COMPLETE") CODE INDICATES THAT THE STUDENT HAS COMPLETED THE COURSE.

THE TWO-DIGIT FIGURE REFLECTS THE NUMBER OF TIMES THE STUDENT HAS COMPLETED THE SAME COURSE.

Figure 11—Two computer managed instruction reports

Assisted Instruction training effort, management is now better able to make the crucial decision: When is each location ready to convert to the new Order Entry System?

MANAGEMENT INFORMATION PROCESSING THROUGH CAI

A major piece of evidence influencing that decision is the information processing gained through Computer Managed Instruction (CMI) reports. At regular intervals, management is provided with the following information:

(1) The number of students (in general) who have completed, or are taking, each course.
(2) The number of students (by location) who have completed, or are taking, each course.
(3) The effectiveness of each course frame—i.e., statistical data on the number of students (and their locations) who err on certain key points within each course. (This is valuable information for the course authors who are continually re-evaluating the clarity of their courses. Information of this kind may indicate a training weakness that is either inherent in the course, or unique to a particular location.)

Since management can monitor the individual training progress of every remote location through CMI, educated decisions can be made regarding each location's readiness for conversion. Continual checking of an Office's progress avoids last minute decisions; management can choose to send special field representatives to help achieve total readiness at a particular location, as soon as a potential training problem appears evident. In this way, management is assured of meeting the planned conversion date for each office.

Although the Computer Managed Instruction reports are most valuable tools, Computer Assisted Instruction itself permits management decisions to be based on much more than just information processing. The crucial decision to convert a location can be made with full confidence in the quality *behind* the information processing data. For example, management can always rely on the following characteristics of Computer Assisted Instruction:

(1) Learning *has* taken place. CAI continually tests and reinforces the material it teaches by asking for practical, interactive responses. Conversely, a procedure manual is incapable of providing individualized instruction, a manager cannot *really* determine that his employee has fully read and comprehended a procedure manual, despite the excellence of the manual itself, until the learning is put into actual practice at the risk of costly errors.
(2) Education has been sound, both psychologically and contextually. Each course was constructed—and reconstructed—by trained experts. The CAI course author, moreover, is continually accountable for his course content and style because his lessons are subjected to scrutiny by those who actually use the system he's teaching. User feedback is both encouraged and sincerely incorporated in all CAI course revisions, as a way of further management accountability to users.
(3) Training has consistently adhered to high standards of style. Although each "computer teacher" has been purposely endowed with a lively and distinct personality, the CAI courses all seem to be part of a larger and more comprehensive training plan. The course consistency eliminates one initial obstacle to sound learning; the adjustment to different styles of teaching and variable standards of expectation.
(4) Training has been customized to the corporation's special needs. Since courses are written *by* Kraftco CAI educators *for* Kraftco employees, there is no danger of the characteristic loss of specificity that often occurs when an outside training package is adapted to one's own needs.
(5) Education is current. Editing of CAI courses can be done on-line or overnight in batch. Late-breaking important changes can be called to everyone's attention through the use of the Mailbox feature of CAI.
(6) Training has shaped attitude very positively. Management recognizes the vital importance of attitude-shaping in any endeavor; since CAI has proven to be one of the best means of creating an outstanding morale among employees, the training program is valued even more highly by management.

FINANCIAL CONSIDERATIONS

Naturally, one of the prime considerations was the cost of implementing the selected training system. An extensive financial analysis took the following factors into account:

(1) Cost of developing an in-house software program versus renting (or rent with option to buy) a CAI system such as the McDonnell Douglas TIME system. The corporation chose to rent the TIME system for a three year period at a monthly rental of $1500. Total cost of the program is approximately $50,000.
(2) Cost of writing the actual curriculum associated with each CAI course. Since one of the major costs in the development of a CAI system is the writing of the course material, the system which is selected must be relatively simple for the course authors to code.
(3) Cost of the extra hardware (if any) which must be allocated to the CAI system. Usually, the extra hardware consists of additional CPU core required over and above the normal transaction processing requirements and additional disk storage space required for the CAI data base.

When these CAI costs were compared to the costs of doing a comparable job of training (measured by the results achieved) using another educational approach—such as field classes or training at headquarters—it was obvious to management that Computer Assisted Instruction is considerably more economical than other training modes. Furthermore, in the opinion of management, cost considerations dictated that obtaining an outside CAI package, and enhancing it, was a better plan than attempting to design a full CAI software system.

CONCLUSION

Although managers of data processing installations have long felt that sound information processing would assure informed decision-making, Kraftco's experience with CAI has convinced its corporate management that good decision-making is *more* than information processing. By its very nature, Computer Assisted Instruction allows the manager to have control of the quality behind the data gathered through information processing, and permits the corporation to place confidence in the role of CAI in the "Fourth Generation" of computer systems. Kraftco's initial experience with CAI has been so successful that Computer Assisted Instruction will be used as part of the company's in-house technical training, as well as for nationwide user education. Once a manager can have confidence in the proficiency level of his employees—a confidence achieved by the skillful incorporation of Computer Assisted Instruction—he can make his decisions with the characteristically sound judgment of the successful executive.

Computer-assisted instruction in industry

by CATHERINE P. BREEN

Montgomery Ward and Company
Chicago, Illinois

The newest training technology is Computer-Assisted Instruction—CAI as it will be referred to here. CAI is an abbreviation for Computer-assisted Instruction; that is, instruction prepared by a human teacher for presentation under computer control. Its primary aim is to *optimize* the learning process.

CAI is worth consideration because, if for no other reason, it is *here*. With so much of business "on the computer," it is only a matter of time before the computer is used for instruction in industry as it is now used in education. Some people believe that industry, because of its pressing need to teach new job skills, will make a far greater use of CAI than academia which, in many cases, is bound by the traditional classroom.

In the United States, most of the work on CAI has been done in federally funded projects in a few large universities. To name a few, Indiana State University, Florida State University, Ohio State University, Pennsylvania State University, Stanford University and the University of Illinois. Computer-assisted Instruction has been called the first transplant in the field of education—likened to an artificial education heart!

How does a trainee in industry learn his job or obtain other kinds of information on CAI?

He goes to the terminal and identifies himself and the program he wants by typing his name and the code for his particular lesson. If I were the student, I would type "Catherine Breen" and if I were learning how to read a financial statement I would type MWFS (Montgomery Ward Financial Statement) or any other code the computer had been programmed to accept.

The computer will check its memory and immediately confirm that it knows me by responding with a code of its own which will permit me to begin my training on the computer.

And, if I were continuing a lesson previously started, the computer would find the appropriate lesson and would permit me to start where I had left off the previous day or previous week. The computer then, becomes my *mechanized, personal tutor*.

The trainee proceeds to read information and answer questions or solve problems given him by the computer, the computer judges his answers, helps him correct his errors, always giving him a second and sometimes a third chance and ends the training by telling him how well he has done—*positive reinforcement* at work!

You can see how valuable this type of training is in industry where training must, for economical as well as practical reasons, be done on a fragmented basis.

There is no need to worry about the inept supervisor or the overbearing, autocratic instructor. The computer is infinitely patient and a personal, private trainer who teaches on a one-to-one basis.

The computer does other things! It can score each employee's answers, it can compare one trainee with others. It can even point up misspellings, if that is important to the learning process.

Another feature of Computer-assisted Instruction is that the instructor is able to obtain a PRINTOUT of the employee's performance to use as a basis for evaluating his progress and the training program itself. The computer will give the trainer a complete report and analysis of a trainee's performance. The instructor can analyze the report and determine which employees need special help. Such an analysis may reveal the real reason for poor performance, a reason that may not be readily discernible in the conventional classroom-like situation. It may not be a matter of "cannot learn" or "will not learn" but may be caused by a poor learning climate or other factors not directly connected with job knowledge. The computer may reveal that an employee knows how to do a job, knows the answers but, for some reason, is not motivated to apply this knowledge. Perhaps he has had a poor "human" instructor!

Now, how does the instructional material get into the computer? How are training programs created for the computer? Keep in mind the program designer is the REAL TEACHER in Computer-assisted Instruction.

The first step *after* determining training needs, establishing short term and long term objectives as well as the target population and initial behavior, is to decide whether the material is appropriate for CAI. This is done by applying the following criteria or judgment factors for determining whether a training program can best be taught by the computer.

First of all, is the information vital to the job or is it just "nice to know?" For example, a program on the electrical circuitry of a trash compactor is vital to the repair man's job but, a program on how many trash compactors have been produced to date and what the outlook may be for their universal use, is "nice to know" information but certainly not vital to his job.

Next, is the information something that must be learned by a *large* population? Do a great number of people need this information to do their job? Montgomery Ward has nearly 500 retail stores spread across the country and it is very important that the 12,000 merchandise department managers in those stores know such things as how to figure turnover and maintain gross profit and understand the fundamentals of merchandising.

Obviously, Computer-assisted Instruction is ideal for this large, widespread population. It can teach critical job skills quickly, and what is *most important*, teach them in a uniform and efficient manner. It will also give immediate feedback so that the instructor can evaluate his training efforts and the understanding and progress of employees.

Another criterion—is the target population small but the subject matter so important and so complicated that it requires the talents of many qualified instructors who are not available at the same time or not available at all? In our case, we have such a situation in training our buyers.

Still another criterion—is the subject best learned on a one-to-one basis? For example, our linear programming of merchandise mathematics in book form is not dynamic enough to hold the attention of Merchandise Department Managers who apparently find this subject too sterile in book form. CAI makes it a viable learning experience through the use of simulation. Interaction between the Department Manager and the computer makes merchandise mathematics come alive. And, the employee can see its relevance to his day-to-day merchandising job. The computer acts as a sparring partner . . . a protaganist . . . the devil's advocate, to motivate the trainee to perform!

So, the computer can be a POWERFUL teaching tool. But, to be effective, the author of the program must know HOW to plan the various types of teaching strategies and know WHEN to use them.

FIRST, CAI must, like programmed instruction, be selective, giving the learner only the information he needs to achieve the training objectives. SECOND, CAI must be versatile, presenting each learner with a sequence of information that is designed to suit his particular learning needs. THIRD, CAI must be self-pacing, permitting the learner to proceed at his own information-assimilation rate.

There are several strategies that can be used to achieve these requirements. The first, and perhaps still the most commonly used in the educational system, is DRILL AND PRACTICE. The next most commonly used strategy or teaching mode is TUTORIAL, followed by INQUIRY, GAMING and SIMULATION.

DRILL AND PRACTICE is a computer presentation of problems, usually arithmetic, which are to be answered by the learner. The objective is to build skills. The learner is given three or four chances to get the correct answer. The program author may provide for an automatic help sequence, that is, a branching sequence, or may merely provide it and let the learner use it if he feels the need. The computer gives the score and stores that score within its record for that learner. A presentation of drill and practice problems such as this is essentially linear with the amount of text kept to a minimum.

The TUTORIAL MODE is very similar to linear programmed instruction. The computer presents a fixed sequence of information with or without branching to specialized help sequences. Information is given, questions asked, the learner answers, and the computer judges. The program author can provide the learner with a help sequence that can be used in two ways. Either the computer can automatically provide the help or the student can request it.

This mode, the TUTORIAL MODE, takes over the main responsibility for presenting a concept and for developing skill in its use. The intention is to approximate the interaction between a patient supervisor and an individual employee. Obviously, an important aspect of the TUTORIAL MODE is to avoid the experience of failure.

In PROBLEM SOLVING, the learner presents quantitative data and asks the computer to solve the problem or, the computer presents quantitative data and asks the learner to solve the problem. In the first case, the computer acts as a calculator. The second is really a form of DRILL AND PRACTICE.

The INQUIRY MODE of CAI is CAI in one of its very best and most efficient forms. The computer presents the learner with a problem, requires him to specify what information he needs to solve that problem, and then provides the information. When the learner has solved the problem, he asks the computer to judge his answer. Rarely will two people ask for the same information to solve the problem. This makes it necessary for the program author to anticipate the facts the different learners require and to build a comprehensive information bank in the computer.

The INQUIRY MODE of CAI is learner-controlled and is the most exciting to me personally and the most challenging both technically and pedagogically. Learner-controlled CAI permits the learner to structure for himself the sequence of instructional experiences he will receive. Since no two learners will follow exactly the same pathway through such a program, it is truly LEARNER-CONTROLLED. However, these programs may be extremely time-consuming to produce and may run six to 10 times as many "instructions" to the computer as does a DRILL AND PRACTICE or a pure TUTORIAL MODE program. But the *learner-controlled program* is worth the effort!

Another type of CAI is GAMING AND SIMULATION. Here a model of some real or idealized complex situation is put into the computer. The learner then must work with and interpret the complex relationship among the variables that represent the situation. The learner puts his solution into the computer where it is compared with the model.

How are GAMING and SIMULATION alike or different? There is a difference although many use the two terms interchangeably.

GAMING . . . the major characteristics are:

1. There need be little or NO similarity to a situation in the *real* world.
2. There is usually a degree of competition.
3. Payoffs are often, if not always, involved.
4. There is an element of "fun," i.e., the learners may not consider the experience as learning.

In SIMULATION . . . the major characteristics are:

1. There is a GREAT similarity to a situation in the *real* world—as great as can be simulated on the computer.
2. A model of "cause and effect" of these events is implied and often explicit, i.e., what happens if "you" do this—or that
3. Transfer of learning is assumed likely as a result of the simulation where the computer presents the work situation as it REALLY is and requires a decision on the part of the learner.

Again, both GAMES and SIMULATION involve the higher-order processes within the cognitive domain—those of analyzing, synthesizing and evaluating information. And, both are LEARNER-CONTROLLED so far as achieving the prescribed objectives are concerned. Industry will use SIMULATION because time and money are involved. GAMING, although *fun*, is not related to the real world of work.

Wards has used SIMULATION in a CAI Specification Buying program where it was necessary to simulate the highly complex buyer's job and give him an opportunity to develop his buying skills ON THE COMPUTER—something we cannot afford to let him do ON THE JOB where millions of dollars are involved!

These are the basic teaching strategies. The program author selects the one or the combination of two or more that best present the subject to the learner. As far as subject matter is concerned, CAI is most effective in the cognitive areas. Affective learning may take place with a CAI subject but most likely it is not BECAUSE of CAI—but merely fallout, so to speak.

CAI programs should unfold in an apparently spontaneous way. To do this, the program designer must plan *possibilities* rather than specific paths. A program designed for teaching with a "let it happen" approach requires a different kind of planning from that used in linear programmed instruction. Industry greater need greater versatility than *that* and can get it if it learns how to use the flexible logic and large random access memory of the computer.

In slide films, Educational Television Programs, audio tapes—the instructional *sequences* and the *time* are *fixed.* In text books and programmed instruction materials, the instructional sequence is *fixed* but the time limit *is not*—an improvement over the former. But, though an improvement, the material is NOT personalized since all trainees or students receive the SAME instruction.

An ideal situation, which CAI can provide, is instruction in which the instructional sequence is NOT FIXED and in which the time is NOT FIXED. A CAI program that has branching sequences is an attempt in the right direction—all branches being different in content but alike in their intent to help the learner achieve an instructional objective.

The next subject is hardware. What types of terminals are in use today? The most common is a teletypewriter—a machine similar to the selectric typewriter and which is equipped to accept paper from a continuous roll. The learner communicates with the computer by typing and the computer communicates with the learner by typing—both communications appearing on the typewriter paper.

A newer, more complex system includes a cathode ray tube which many CAI systems use. This unit looks like a small television set connected to a keyboard very similar to a standard typewriter keyboard. The learner communicates with the computer by typing—the typing appearing simultaneously on the cathode ray tube. The computer communicates with the student by flashing information on the cathode ray tube.

The very newest type of terminal, one developed at the University of Illinois, has a Plasma Display Panel which will no doubt eventually replace the cathode ray tube for Computer-assisted Instruction terminals. The reason? The relatively simple structure of the Plasm Display Panel will greatly reduce the cost of communication between the terminal and the computer. It will bring the cost of CAI down to a point where CAI will not only be feasible but economically practical in industry.

We, at Montgomery Ward, feel privileged to have PLATO terminals and to be part of the University of Illinois' research project on the PLATO CAI system. We are indebted to Dr. Donald L. Bitzer, Director of the Computer-based Education Research Laboratory at the University of Illinois and co-inventor of the Plasma Display Panel, for the opportunity to research the PLATO System for industrial training.

Dr. Bitzer will, through his invention, make education available to large masses of people, who up to this point have had little or no access to learning. In industry, we anticipate that CAI will also provide knowledge and skills to a great number of minorities who at the present time occupy low level or entry jobs with little or no opportunity for upward mobility.

Wards, in addition to the PLATO System, is a user of the IBM Interactive Training System which uses a cathode ray tube terminal. Programs have been created for training in the operational and merchandising areas on both systems.

Back to the Plasma Display Panel. Instead of the cathode ray tube, PLATO, as mentioned before, uses a Plasma Display Panel. This device consists of two thin layers of glass between which is a rectangular array of small neon gas cells. Any cell can be selectively ignited to form part of a word, diagram, line drawing, graph, etc. Data arrives at the terminal from the computer via a voice grade telephone line.

The Plasma Display Panel is approximately twelve inches square and contains 512 digitally addressable positions along each axis. Pictures from the terminal's random access image selector can be projected on the display panel and computer-generated information superimposed on those pictures. This combination of static and dynamic information makes complex displays possible. The image selector's film plate can contain up to 256 images and can be easily inserted and removed by the learner.

When the terminals are outside the computer center, and this in all probability will be the most economical and practical use of CAI in industry, they will get their signal from the computer by voice grade telephone lines.

By the middle of 1974, we will have ten PLATO terminals in our Corporate Office in Chicago. Eventually we plan to share a coaxial cable or microwave installation with universities and other institutions in the Chicago area. This will greatly reduce data transmission costs. Right now there is a cost of approximately $120.00 per month per terminal for the telephone line to the central computer.

In addition to data transmission costs, there are rental and management services costs PER TERMINAL charged us by the University of Illinois. These are approximately $2500 per terminal per year.

At Wards, all training programs which are put on the computer reflect both short term and long term corporate goals. Because Computer-assisted Instruction insures consistency and objectivity, Wards plans to use this computer talent to make certain that *training* goals reflect *Company* goals. All training in industry, no matter what technology is used, must have as its prime purpose the improvement of employee skills and bottom line performance—an *increase in profit*!

Some of the CAI programs we have are:

Turnover . . . This is a merchandise training program for the Retail Department Manager which shows him the effects of sales on inventory, inventory on sales, and gives him an opportunity to experiment with sales and inventory figures so that he can LITERALLY see the effect on his merchandise turnover. For example, if the Department Manager wants to increase his turnover, he will find that he must increase his sales if he keeps his inventory the same or will have to reduce his inventory if his sales remain the same as budgeted. The computer quickly calculates the turnover based on the trainee's input—a simple but meaningful application of the PROBLEM SOLVING teaching logic.

How to Read a Financial Statement . . . It is imperative that our buyers understand the balance sheet in order to select the best manufacturer for our merchandise.

Determining Price Points . . . One of the things a buyer must do when planning a line is establish promotional selling prices as well as basic selling prices. This program lets the buyer trainee experiment with different price points and shows him the effects of his pricing on his total profit goal.

Return on Merchandise Investment . . . This is a program designed to help the Retail Department Manager understand the effects of turnover and maintained gross profit on the total profit goal.

Merchandise Mathematics . . . As I said earlier, this was formerly a P.I. program which was too lean—it did not have enough practice problems for most people. The CAI version uses SIMULATION to relate the mathematics to the real life situation. We also use DRILL and PRACTICE problems—perhaps one of the fun instances where we will use that teaching mode.

Right now, we are going to show you portions of two Montgomery Ward programs on PLATO IV and two or three programs developed by the University of Illinois.

Montgomery Ward Specification Buying . . . This is a program for the buyer trainee which teaches him, through the TUTORIAL and INQUIRY and SIMULATION modes, how to develop performance and technical specifications for a product. In this program, we have used a portable, electric drill as the product.

Montgomery Ward How to Repair a Trash Compactor . . . The trash compactor is a new product which compacts trash into a relatively small space—a product developed because of our country's need for pollution control. Since the product is new, it is important that our service repair men all over the country know how to install and repair it. In this program we have used the great capability of PLATO IV to show colored slides and superimpose computer-generated material on those slides. For example, the trainee is asked to identify the controls of the compactor. He is shown a picture of the control panel, and superimposed on the picture of the controls is a computer-generated question which asks him to list the types of controls.

University Programs . . . First, we will show you a course in College Genetics to illustrate how the computer can simulate a testing laboratory. Then, we will show you a number of designs that have been "drawn" by the computer.

The above programs were selected to illustrate the teaching, slides and graphic capabilities of PLATO IV. The complete PLATO IV System also has an audio attachment and a touch panel, both of which are accessed through the computer program.

Let me emphasize that as a training director using PLATO IV, it was not necessary for me to have prior knowledge of a programming language or computer technology.

The PLATO IV System uses the TUTOR language a simple, pseudo-English programming language which permits a trainer to "type" his lessons directly into the computer.

The trainer can compose, edit, test lesson material as well as analyze student responses using the same terminal the learner uses.

No traditional computer program need be written—no systems programmer need be employed. All the trainer or would-be trainer needs is his own well prepared lesson plan containing all information to be presented and all answers to problems and questions to be presented.

Any trainer or would-be trainer who is thoroughly familiar with his subject could, knowing the TUTOR language, type his material into the computer *without first having put it on paper!* For example, a Comptroller at Wards could teach payroll control, a Merchandise Manager could teach merchandising replenishment, a Field Auditor could teach retail auditing, a Corporate Office training staff member could teach management skills.

Wards is conditioned to the use of Computer-assisted Instruction. About six years before working with PLATO, Wards designed a Decision Making CAI program for Retail Store Management Staff using SIMULATION as the teaching mode. However, they were restricted to the use of the teletypewriter and a computer mathematical model. In this Decision Dynamics program, two teams of retail store management staff put into the computer, their "merchandise-operating expenses" for each month of the two six-months periods of the program's fiscal year.

Figures are then compared with the ideal model stored in the computer and the participants told via the resulting printout whether they have made a profit and if they have met their goals for that year. Now, that Wards has the versatility of CAI, it plans to use case studies, role playing and other teaching modes which will more clearly illustrate the real merchandising situation in retail stores.

Once CAI systems are exposed to Management, there will be many demands for training programs and there will be a temptation to put EVERYTHING on the computer! However, it is important to resist that temptation and apply the criteria presented here.

Presently bottom-line uses of CAI for are:

- *A Comprehensive Buyer Training Program* covering such critical areas as Analyzing the Market, Selecting a Supplier, Contract Buying, Implementing the Buying Plan, etc.
- *Retail Department Manager Training* including Sales and Inventory Budgeting, Inventory Control, Operating Procedures, Promotional Planning, Determining Basic Needs, and Management Skills.
- *Sales Personnel Training* to include cash register and salescheck training, product knowledge, customer service, and handling of refunds and exchanges. Salesmanship is not included at this time; it is not certain that the computer can do a better job than a "live" instructor! It is important to reserve judgment as to the effectiveness of computer training in the Human Relations area.
- *Retail Store Manager and Staff Training*—Here we anticipate a real breakthrough and hope the computer can shorten the orientation period for these critical personnel. I wonder how much time and money is wasted while middle management, and even TOP management, familiarize themselves with the complexities of their new positions?
- *Basic Skills Training*—These will cover the knowledge areas necessary for minorities to attain upward mobility and career ladder planning. Included are basic English, arithmetic, human relations and supervisory training.

Much is being said today about ORGANIZATIONAL DEVELOPMENT and a lot of time and money is being spent on programs to create the kind of climate that will allow employees to operate more efficiently and more courageously. Basic to the concept of ORGANIZATIONAL DEVELOPMENT, is the security created by a knowledge of job skills. Computer-assisted Instruction will be a strong partner in the area of ORGANIZATIONAL DEVELOPMENT!

Trainers in industry must be constantly alert to new developments. SECONDLY, education and industry *can* learn from each other. Education for many has lost its mystique. Young people regard it as a "necessary evil" to enter the world of work—a "tribal rite" which earns them a license to hunt, fish and procreate! They look for short term goals, early responsibility and what is MOST important, feedback. Any educational experience must be relevant and simulate more accurately than it presently does, those problems encountered in daily living. Hopefully, CAI will act as a communicator—as a sharing mechanism—which will enable both industry and education to satisfy these needs!

The use of CAI at McDonnell Douglas

by BERNARD H. GODDARD

McDonnell Douglas Automation Company
Seattle, Washington

Computer Aided Instruction (CAI) gives you increased productivity and reduced training costs. Increased productivity is imperative in today's economic environment—people must be trained to perform at top quality levels in order to maintain your high productivity. We have done studies at McDonnell Douglas on training costs with both a control group and a test group using identical course material to find out the training costs involved in a conventional classroom situation and in a CAI presentation. Training time can be reduced with CAI—in some cases as much as 50 percent—over conventional classroom training. CAI will give you increased productivity by training your people better, faster, without social disorder, in their assigned area, on the equipment they're going to use, and by reducing the training costs—both by the method of presentation and the lack of travel involved.

In our studies at McDonnel Douglas it was determined that a definite cost reduction was afforded through the use of CAI. Even though McDonnell Douglas owned the TIME software package, the corporate office did their own study with the following total cost results on two types of courses.

Technical Training	Conventional Classroom	TIME System
100 students	$11,354.	$7,128.
200 students	$21,728.	$12,094.
300 students	$32,102.	$17,060.
Cognitive Training		
100 students	$2,182.	$2,394.
200 students	$4,052.	$3,973.
300 students	$5,922.	$5,552.

The training time was reduced by 53 percent in the technical training and 23 percent in the cognitive training.

Let us consider an old idea that we want to get away from, like the one-room schoolhouse. Many classes taught even today are this variety, where you have a class of students in a classroom, and one instructor. The instructor must get the information over to the trainees—what's the best way to do that? The best way is to have one instructor, one trainee; therefore you have a one to one training relationship. You DON'T have that relationship in the one-room schoolhouse approach because in the one-room schoolhouse, everybody's getting the same material whether they need it or not. Can you afford this? If you have one instructor with one trainee, you have maximum training of personnel. But the costs would be very high. How do you do this economically? How do you get the information to the trainee and the trainee's questions answered all on a one-to-one basis?

What happens in the usual classroom when one person in that group asks a question? The question ties up the other people in the whole class, but maybe it relates to only four or five people in the entire group. What are the rest of them doing? They lose interest and you have lost your concentrated one-to-one training. Or, on the other hand, if everybody asks a question at once, CAI can still give you this one-to-one relationship, by dealing with each question as it turns up. The instructor using CAI has the medium to train each trainee with one-to-one teaching. He develops the course, writes it, presents it, through the computer to the terminal to the trainee. The instructor becomes many instructors to many trainees, since each one sees the terminal as the instructor. The trainee using CAI has the ability to gain access to the instructor through a message function and is using a terminal that in most cases he will be working on in his job. The trainees see the instructor as being copied so each student is talking to his own instructor on a one-to-one basis, and has, so far as he is concerned, a separate instructor for each course.

Now, where can we use this type of training? In today's environment, you can use this in almost all applications because today's technology requires that your trainees must know the facts.

At McDonnell Douglas CAI will be used in all divisions of the company to train technical skills and cognitive data. In some cases the Management Development training will use CAI and the conventional classroom in conjunction to train their personnel. It will greatly reduce the training time that is now required for the trainee to spend in a classroom.

Three way instruction control using the computer, the instructor and the trainee can control the course by the following. The trainee can review the course at any time he wishes; he may request instructor assistance, or he may continue the course. The trainee can go step by step, straight through the material, or he can stop, and continue at a more convenient time. The trainees at McDonnell Douglas were most receptive to the CAI approach. They liked the idea of the freedom of CAI versus the structured classroom regimentation. With CAI the trainee may miss an assigned time period and still not miss the training material. If they are sick or work builds up to the point where they must get it done then they can do both, the training and the other (work or sickness) without missing out, and the main point is that the employer will benefit as well. The training that is needed is obtained at the most convenient time and work schedules can still progress.

The instructor can build in review positions at whatever point they are needed throughout the course and then the computer will control these reviews. The computer controls what's going on in the following manner: it monitors the trainee's progress and the time that it takes him to go through the course. It tailors trainee lessons according to how the instructor has set up the course at the time it was written. The trainee may be required to go step by step through a series of functions that he must prove he can perform. Or, if the trainee proves he can do a certain step correctly, he goes into some new type of material, and the computer controls that. The computer executes trainees' requests—they may take different courses, they may stop courses, they may continue courses, etc.

The computer can also monitor course effectiveness and with this data the instructor can check to see whether the course is providing the material the trainee needs, or whether the trainee is getting the type of material that is going to improve his job knowledge. By going through the statistics kept automatically by the computer after a course has been completed, you can get an item analysis immediately from the questions, you can get the path that the trainee took through the course, and you can also have instructor/trainee communication, if needed, through a message function. So now the instructor can look at the course, evaluate it, change it, communicate with the trainee, evaluate him, and monitor the progress of all trainees. The instructor can determine whether or not the trainee has obtained the information that is needed, and whether or not the course did what it was supposed to do. If it did not, the instructor may change the course. You can evaluate the trainee, determine whether or not the material was valid by the pre-set test standard, and whether it has provided the necessary skill to the trainee.

You can monitor the progress of the trainees very simply. I think Jim Adams of MCAIR did a fantastic job with his students, when they were taking the course on Binary Numbers and Boolean Algebra. Jim would dial into the computer each morning with the name and ID of each student, and that would tell him exactly how far that trainee had progressed the night before. Jim knew exactly how far each trainee went each day and if any were having trouble without ever being there to actually see what they did in their progress and he could communicate with the trainees if he had a need. The instructor has a lot of control with CAI due to the record keeping of the computer. Jim was also released from that group of students and he was available to MCAIR to teach a conventional on-hours course. We did some comparison studies at the Corporate level on retention within MCAUTO. We found that the retention rate under CAI was about 5 percent higher at the completion of a course than with a conventional classroom presentation, using a control and test group. Also, CAI post test results showed about 5 percent improvement over conventional classroom training after 90 days. The retention rate also remained higher with CAI than with conventional classroom instruction. The study at McDonnell Douglas had the following retention figures at the completion of the training.

	Conventional Classroom	TIME System
Technical Training	88.26%	93.06%
Cognitive Training	75.0%	76.4%

A 90 day post test was given with the following results:

	Conventional Classroom	TIME System
Technical Training	85.3%	90.0%
Cognitive Training	63.75%	65.29%

Therefore the results show a definite advantage with CAI over the conventional classroom.

Skill levels can be changed, by determining what skill level is required and then training to that level, testing trainees to assure that they are at that level, and then testing periodically to see if they still maintain it. If they have trouble maintaining the level, retrain them and maintain them at that position. CAI training is faster because the trainee will progress at his own rate. In a conventional classroom, all trainees are going to progress at the rate of the slowest individual in the class. In a conventional classroom you have to teach to the slow students because today's technology requires that you have full knowledge of the material that you're going to be using. JCL for example, might be an example of a situation where one could get by with minimum knowledge (If you know what the JCL is, you can run a program very simply and correctly.) but that person would have to rely on someone else's ability to check his JCL level.

You can measure trainee proficiency to handle job demands. You can simulate the job the trainee is supposed to do wherever he is employed. If your operation covers a large geographical area, you can have the trainees remain at their position, with courses available 24 hours a day instantaneously without moving people from place to place. One of the biggest costs of training today in big companies is sending people from one location to another to train them on the various aspects of the job.

Some CAI systems require that you be a programmer to use them while with a few you do not need to know how to do programming. Some programmers can write training courses, but usually programmers do not understand educational objectives, or are not interested. If your field is training, and you're trying to train, then you need to be able to write courses without using a programming language, so that both kinds of trainee can understand your courses.

In McDonnell Douglas's TIME, you may use straight English or any language and write a course the way you think you want it to look, have it keypunched or entered from a terminal into the computer and in either case, you can see exactly the way the presentation will appear to the trainee. If it's going to be presented on a CRT terminal, look at it on a CRT; if it's going to be presented on a hard copy terminal, it should be looked at on a hard copy presentation. In any case, you can get an idea of what the course is going to look like instantaneously. Course appearance is definitely an enhancement of your training and of student acceptance of CAI.

CAI can be presented with various types of courses; tutorial, drill and practice, test, simulation and gaming, and special features. Tutorial courses usually are used to present new material. The trainee is given information on one objective and then a question is asked. The trainee replies and a new segment of information is presented based on that reply. The trainee does not move to new material until he has mastered the current material. CAI is most effective because of the concentration of study plus the response that the instructor builds into the course based on the trainee reply.

With drill and practice courses, a trainee is drilled on the subject matter for as long as necessary. The trainee may also use practice problems from his own area. You may give all trainees the same or different material. With CAI, the computer keeps the test scores, the accuracy of each trainee, what test results he had, what answers he gave, what path he took through the particular training course. Those test scores can be measured against a pre-set level so that the trainee must maintain 90 percent or 70 percent or whatever the score you set. When you write the behavioral objectives to the course, you will state what level of skill, or what level of achievement the particular course requires in order for a trainee to pass.

In simulation and gaming you can set up various situations, either actual or imagined and have the trainee interact. It may be the completion of forms or the correction of errors related to a job. Games can also be played with CAI, to generate thought processes or just to play games—some managers believe the games could be used as a reward after a trainee has spent time on training courses.

Special features consist of a catalog of courses which give you a hierarchy of all courses within the system. The catalog does not need to be limited to just courses, it may be a hierarchy of any type of material or information you want to list. For example: suppose we wanted to take our written procedural material specifications and place it in a hierarchical tree so that we could have easy access to them. Or, what we know with regard to aircraft engines can all be cataloged under one branch of the catalog in a hierarchical tree. Any type of system or material that you want can be placed in a hierarchical order. The reference library consists of all the material that you want to put in a description of, what the various courses cover, and whom they relate to.

The management portion allows you to obtain management reports, on line text editor, and course generator. Under course generator you can generate new courses either on-line from a terminal or in batch mode through a batch reader. Course generator gives you a listing of all course material that has been generated line by line. In on-line text editor you can add a frame, add a line to an existing frame, change the destination of a frame, change the frame number, correct spelling, or change any other portion of the frame contents that need to be modified. Management reports allow you to look at the status of each trainee. Statistics available by the trainee are, 30 completed courses, five active courses, trainee name, trainee id, course name, course id, and path through a course. Course statistics are kept on how many people selected each answer within a frame to provide you with reliability of course material, course name, course id, and course size.

In Summary CAI offers increased knowledge levels, reduced training costs, no travel of either trainees or instructors, no disruption of work schedules, and higher retention of course content than is found with conventional classroom training. The instructor can instantly monitor all courses, he can evaluate course content, and correct or modify it as needed. Also if you have multi-time zone coverage, the material is always available, 24 hours a day when your computer is available. Probably one of the outstanding features of CAI is the equal quality instruction and management at all locations. Any time that you develop and present a conventional course, it's always best the first time—(you're tuned to a peak when you get ready to teach it), then you kind of let down a little bit each successive time you teach it, you kind of have the tendency to say "well, I've taught this before, I know it, so I don't have to worry". With CAI that never happens because if you find something new, you add that to the course, and you keep adding—you may get rid of some deadwood as needed, but you're only adding to course content and therefore enhancing the level of instruction. In management you have the capability of all locations having the same course; this is not the situation when you have sent 10 instructors into the field with 10 different ideas of what a particular job might be. With CAI there is only one description and it's always the same. Reduced travel expenses: almost no travel expenses involved in CAI courses. People get their training in their own work area—one of the big values of CAI is no social disorder, so you don't have the problem of upsetting car pools, and forcing people away from their normal work time, and place.

PANEL SESSION PAPERS AND PAPER ABSTRACTS

A panel session—Intelligent terminals—Rationale and implications

SESSION CHAIRMAN—L. C. HOBBS

Hobbs Associates, Inc.

Panel Members

L. C. Hobbs—Hobbs Associates, Inc.
Dr. Marcian E. Hoff, Jr.—Intel
T. B. Steel, Jr.—Equitable Life Ins. Co.
Charles R. Fisher, Consultant
C. W. Rosenthal—Bell Telephone Laboratories

OVERVIEW—L. C. HOBBS

Continuing reductions in cost of LSI logic and memories is making it increasingly attractive from an economic standpoint to include processing and computing capability in terminals. At the same time, communications costs are remaining constant or decreasing at a much lesser rate. Therefore, maintaining a properly balanced telecommunication system requires distributing, processing, and computing functions to the terminals in many cases. A number of manufacturers are now offering smart or intelligent terminals which include varying degrees of processing and computing capability which can be used to handle part of the users processing and computation tasks and to reduce the load on the communications lines and the central processor. On the other hand the economy of scale still favors centralized mass storage and certain types of applications require a central data base to permit multiple users to access and update the same large files.

This session will consider the interrelations of the different parts of a telecommunication system which include intelligent terminals. Questions such as which processing and computation function should be performed in the terminal and which in the central processor, type of system software required to allocate tasks between the terminal and central processor, and the types and capabilities of peripheral equipments that should be incorporated in the terminal will be addressed. Hardware, software and communications technology trends and their impact on intelligent terminal systems will be discussed. User requirements and problems will also be considered. In addition to brief presentations from each panelist, ample time will be allotted for discussions and audience/panel interaction.

HARDWARE IMPLICATIONS OF INTELLIGENT TERMINALS—DR. MARCIAN E. HOFF, JR.

Developments in LSI technology have made the intelligent terminal a reality. Further developments in LSI technology will allow ever increasing amounts of computing function to be delegated to the terminal. It is not unreasonable to expect compact terminals to contain processing power exceeding that of today's medium scale computers, and to have sufficient memory for all but the largest of jobs. The typical central facility may become more library than processor. Development of low-cost high capacity file storage units could even further decentralize computing function.

SOFTWARE IMPLICATIONS OF INTELLIGENT TERMINALS—T. B. STEEL, JR.

The implications of intelligent terminals on software design and development range from trivial to profound depending on the degree of "intelligence" in the terminal. At one end of the scale the ability to perform minor editing at the terminal merely relieves the central processor of a task. At the other end of the scale the problem begins to merge with those of computer networks.

This discussion will concentrate on the middle of this spectrum, considering the implications for modularization of operating system and data base management systems as well as a not very clearly understood impact on programming language design.

COMMUNICATIONS IMPLICATIONS OF INTELLIGENT TERMINALS—CHARLES R. FISHER

Two important changes in the communications industry protend lower cost and higher speed data transmission. The change to digital techniques for voice circuits will probably take place as the cost per voice circuit remains nearly constant in 1974 dollars. Thus, the tariffs for voice circuits should stay nearly constant in 1974 dollars (or perhaps even reduce slightly due to competition from Specialized Common Carriers). However, digitized voice circuits use 64 kilobits to achieve each voice channel. Perhaps as much as ½ the channel bit capacity needs to be allocated to overhead and other costs of operating the channel as a data channel instead of a voice channel. This would in turn give 32 kilobit data service. At the present cost of a voice channel, the advent of high performance and responsive packet and/or message switching systems will allow the user to utilize the communications system on a demand basis. The user will benefit from this economically through tariffs which tend to charge by the bit rather than unit of time, eliminating charges for unproductive circuit utilization incurred with current circuit switched arrangements.

USER IMPLICATIONS OF INTELLIGENT TERMINALS—C. W. ROSENTHAL

The so-called "intelligent" terminal is used as an adjunct to a central maxi-computer where it carries on significant logical functions. It is also used in stand-alone configurations where it intermittently or infrequently connects to a central computer to acquire a program, load data or return data. These terminals are a solution which has found the right problems. The solution is composed of low-cost mini- and midi-computers and their peripherals which are plausible because of integrated circuit technology and intense competition. The solution is also affected by the increasing understanding of computers by users in labs, schools and plants.

Problems solved by using terminals in place of central computers include: protracted delays in providing centralized time sharing of adequate cost, accessibility, resources and reliability; changing central computer software effects on stable programs which are used over and over again; lack of personal control on priorities and schedules; and interference among users causing loss of privacy or loss of work when others crash.

The considerable advantages of terminals are somewhat balanced by disadvantages such as: the administration of purchase and maintenance contracts; heavy involvement in software and hardware technology which may be remote from one's fundamental discipline; dedication of space; coupling to obsolescent equipment, and unwitting replication of stand-alone terminals beyond the point where a central computer can serve many users.

A panel session—The effect of changing technology on computer graphic systems

MODERATOR—PHILLIP P. DAMON
Hughes Aircraft Company, Industrial Products Division

Panel Members

James D. Foley—University of North Carolina, Department of Computer Science
Arthur D. Hughes—Hughes Associates
Carl Machover—Information Displays, Inc.
Sol Sherr—North Hills Associates
Robert H. Stotz—University of Southern California Information Sciences Institute
Luis Villalobos—Hughes Aircraft Company, Industrial Products Division

OVERVIEW—CARL MACHOVER

Rapidly changing technology is affecting every aspect of computer graphic systems. Basically, a computer graphic system consists of an interface/display processor, function generators, display, operator input/output devices, and software. Each element will be briefly described as an introduction for subsequent speakers who will discuss technology impact on the elements. Tradeoff considerations among the various elements will also be discussed.

INTERFACE AND DISPLAY PROCESSOR—ROBERT H. STOTZ

Today the interface between graphic display generators and application software comes in many different forms, ranging in speed from phone lines to megabit parallel channels and in performance from simple code translators to full blown programmable-processors. With the ever-lessening cost of processing power, the philosophical questions of what should be done in the display and what should be done in the host computer deserve review. Several new graphic display products will be discussed in this session as illustrative examples of the approach being taken by various manufacturers. In addition there is mounting activity toward standardizing graphic communications, at least for the ARPA Net. The status of that activity will also be reviewed.

GENERATORS—LUIS VILLALOBOS

Graphic generators convert digital data to visual images. This discussion is limited to line drawings; it excludes generators for solid areas, edge, fill, and symbols.

Early generators produced all images out of dots by placing the dots close together (roughly 100 points/inch) to produce smooth-looking lines and curves. The data required is proportional to the total image arc-length; hence, scaling affects either image quality or data requirements.

Vector generators draw straight lines, from the previous vector end point to the new point specified. Vectors reduce data from 100 points/inch to 1 point/vector, regardless of size.

Vectors can be used to approximate curves, but about 20 vectors/inch are required to obtain smooth curves.

A "good" curve generator does for curves what vector generators do for straight lines, i.e., significantly reduce data required for curves. Curve generators draw specific curves, e.g., conics, exponentials, and circular arcs (from the previous end point to the new point specified). The most generally useful generator curves appear to be conics. A conic generator reduces data for curves from 20 vectors/inch to the equivalent of 4 vectors/curve, regardless of curve length. Curves such as spirals or curves with inflection points require more data.

DISPLAYS—SOL SHERR

The majority of presently available computer graphic displays are of the refresh type using random deflection, or vector CRTs. These offer full graphics capability and considerable flexibility in image formation and presentation. The digital television refresh CRT systems, while very common as alphanumeric displays, have been largely restricted to limited graphics for economic reasons. The direct view storage display offers a full graphics capability at very low cost by incorporating storage in the CRT, allowing low-speed data transmission and generation electronics. The electrical storage system has most of the same characteristics plus zoom and selective erase capability. It improves the visual image by using standard television monitors.

There are only two types of flat panel or matrix displays commercially available, both based on gas discharge phenomena. These are the AC gas discharge or plasma panel which has been incorporated in a few computer graphics systems, and the DC gas discharge which is used primarily as an alphanumeric unit but has shown the ability to create limited graphics images. Other flat panel displays using liquid crystals or light emitting diodes are still in the early stages of development.

No system appears to be in a position to oust all of the others at present. The ideal system is still to be achieved and remains a challenge for display designers.

OPERATOR INPUT/OUTPUT DEVICES—ARTHUR D. HUGHES

Discussions of computer displays often concentrate heavily on transfer of information to the operator from the system, sometimes losing sight of the need to transfer information and control from the operator to the system. This reverse transfer process using anything but the hands of the operator has not yet been fully developed. Most designs presuppose some interactive operation between human and system. This operation occurs in concert with an output signal back to the operator to acknowledge his action, usually through the eyes of the operator.

Existing I/O devices are found using two primary techniques, one with operation directly in an x-y plane (light pens, tablets, digitizers), the other not (controls, switches, keys). Either set of techniques may be associated with the display screen, except that the light pen, expected to remain popular, must be. However, touch-sensitive screens allow the operator simply to use the finger or a stylus.

Looking into the future, suppose we consider use of the operator's voice and hearing instead of hands and eyes for operator input and output. There are already devices on the market for voice input particularly associated with computer audio response systems. However, I believe that voice input, with or without voice answer back, could be an important addition to the list of operator I/O devices for displays.

SOFTWARE—JAMES D. FOLEY

Display architecture impacts graphics software in at least two ways: ease of programming and machine-independence. It has become axiomatic that if computer architects and computer programmers collaborate in conceiving a new computer, the result will likely be more versatile and powerful than otherwise. With one or two notable exceptions, display system architects have not applied this axiom to their own endeavors. Consequently, most displays are not programmer-oriented, making the writing of graphics system software needlessly complex.

The second area of impact is machine-independence. This is the ability to move a graphics application program from one display system to another. There are numerous problems, yet the capability is being sought by many users. Displays cause difficulty both at the broad architectural level and at the level of details. Differences in basic architecture will and should always exist. But on the contrary, low level details such as character codes, or character spacings, coordinate system, and line structure types are rather unimportant and could well be standardized. This would eliminate many programming problems of machine-independence.

A panel session—Mass memory systems

SESSION CHAIRMAN—JOHN C. DAVIS

Department of Defense

Panel Members

Dennis Luck—Department of Defense
Eric Salbu—Ampex Corporation
Carol Peters—Informatics, Inc.
Robert Koenig—Control Data Corporation
Glen Bacon—IBM Corporation

OVERVIEW—JOHN C. DAVIS

The concept of what capacity constitutes a "mass" memory system has been growing by 10^4 bits per decade. This trend is expected to continue at least through the 1980's. A mass memory system as now defined must have a capacity of greater than 10^{11} bits.

What technologies, architecture, and software are used in current mass memory systems? What is the cost of ownership of a mass memory system? Who needs mass memory systems? What are the prospects and projections for new technological innovations in the field of mass memories?

In this session the answer to these questions will be addressed with the emphasis placed on establishing the current state of the art in both mass memory hardware and software.

MASS MEMORIES—DENNIS R. LUCK

A brief review of the state-of-the-art of third level mass storage systems will be presented. The general thrusts of mass storage developments now underway will be outlined. Mass storage cost benefit analysis considerations, system integration requirements, storage networking trends and user benefits will be included. The evolution of the storage hierarchy will be discussed.

MASS STORAGE SYSTEM IMPLEMENTATION APPROACHES—ERIC SALBU

The optimal Mass Storage System (MSS) Implementation/Interface approach selected is heavily application dependent, e.g., response time versus throughput requirements are different for real time and batch.

The characteristics of the Host System and the Mass Storage System will also influence the total implementation approach, both with respect to Hardware and Software. The presentation will address this topic in a somewhat generalized manner.

SOFTWARE REQUIREMENTS FOR MASS STORAGE SYSTEMS—CAROL B. PETERS

Mass Storage devices provide a new level of on-line storage. This level is at present characterized by large storage capacity, slow access, good transfer rates, high density recording, and relatively low cost. The characteristics of the devices are such that they are being imbedded in systems which include controlling mini-computers. The entire hardware/software system is referred to as a Mass Storage System.

Mass Storage Systems provide a new type of data residency—one which is a hybrid of on-line data residency and off-line library storage. Data which is stored on a Mass Storage device is on-line, but it is not as readily accessible as data stored on fast access devices. The large capacity of Mass Storage devices and low recording media costs make it practical to retain on-line large volumes of data formerly retained in tape/disk libraries.

This paper discusses the software requirements for Mass Storage Systems. An overview of current software developments in the Mass Storage Systems area is included. This is followed by a discussion of future requirements and a discussion of data flow through storage hierarchies/networks.

THE SCROLL MASS MEMORY SYSTEM—ROBERT KOENIG

This paper describes the SCROLL Mass Storage drive under development by Control Data Corporation. The SCROLL drive combines a rotating head with a controlled foil bearing to implement non-contact recording on a wide web of magnetic tape. The tape handling mechanisms provide for precise sensing and control of tape position, and are implemented to prevent any contact with the tape medium. Tests conducted to date show the level of recording reliability is equal to that of discs. Data formats and hierarchy are the same as those used on discs. The capacity of a single SCROLL drive is in excess of 100 billion (10^{11}) bits.

TECHNICAL EVALUATION OF MASS STORAGE SYSTEMS—GLEN BACON

As Magnetic Recording Technologies and the several alternate technologies improve, the range of possible cost/performance tradeoff in mass storage will increase. This will allow new configurations for mass storage devices, as well as influencing the hierarchy of other storage devices which support the system. This paper will develop assumptions to support improvement trends in the several technologies and access cost/performance possibilities for the major alternatives. Points of technology development at which the relative order of the alternatives change will be identified.

A panel session—Current trends in the software products industry

SESSION CHAIRMAN—MARTIN A. GOETZ

Applied Data Research

Panel Members

L. A. Welke—International Computer Programs, Inc.
Patrick McGovern—International Data Corporation
Burton Grad—IBM Corporation

OVERVIEW

This session will examine the Software Products Industry, concentrating on the economics of the industry, the problems facing the software manufacturer, and the changing needs of the user. The panel presentations and discussion will be based on the following: (1) the industry is rapidly growing both internationally and domestically, with U.S. revenues expected to exceed $1 billion by 1976; (2) the industry's expansion is forcing the software manufacturer to face new problems in developing, maintaining, distributing, supporting, and marketing his product; and (3) the types and sophistication of software packages available to users are constantly growing and changing.

A panel session—Applications and extensions of the TENEX operating system

SESSION CHAIRMAN—JERRY D. BURCHFIEL

Bolt Beranek & Newman, Inc.

Panel Members

Mel Pirtle—NASA AMES Research Center
Edward Fiala—Xerox Palo Alto Research Center
Robert Thomas—Bolt Beranek and Newman, Inc.
Daniel Murphy—Digital Equipment Corporation
David Walden—Bolt Beranek and Newman, Inc.

OVERVIEW

The TENEX system was developed in 1969 to serve as a powerful, flexible, yet inexpensive research facility. It provides virtual memory, a hierarchy of processes within each job, a pseudointerrupt system for interprocess communication, and a highly human-engineered command language. This system has become such a popular research tool that there are (as of January 1974) 12 TENEX systems in operation. Ten of these TENEX systems are hosts on the ARPANET, a national computer resource sharing network developed by the D.O.D. Advanced Research Projects Agency.

This session will explore interesting aspects of the computing environment provided by TENEX, emphasizing the new facilities and modes of resource sharing which TENEX provides.

A panel session—A large real-time system development

SESSION CHAIRMAN—M. P. FABISCH

Bell Telephone Laboratories

Panel Members

N. H. Brown—Bell Telephone Laboratories
J. W. Olson—Bell Telephone Laboratories
W. S. Doyle and J. R. Gibbons—Bell Telephone Laboratories
J. P. Haggerty—Bell Telephone Laboratories
B. P. Donohue, III and J. F. McDonald—Bell Telephone Laboratories
R. R. Conners—Bell Telephone Laboratories
R. D. Freeman—Bell Telephone Laboratories
H. M. Jackson, II—Bell Telephone Laboratories

OVERVIEW—T. H. CROWLEY and N. H. BROWN

Development of the large, real-time data processing system for the SAFEGUARD ballistic missile defense system has led to many innovations in both hardware and software. The objective of this and the following talks is to impart some of the lessons that we have learned, which we believe will be useful to the data processing community.

The system must achieve very high throughput and reliability. To help meet this objective, a multiprocessor architecture has been employed.

The multiprocessor and other system characteristics have created new challenges and developments in operating systems and applications software development.

Extensive testing has been required to achieve high confidence in correct system performance. This led to a new approach, in the use of redundant hardware for simulating test inputs.

To support this large development effort, numerous commercial computers and service programs were used. New methods for debugging, integrating, improving software quality, and controlling changes were sought.

ARCHITECTURE OF THE CLC COMPUTER—J. W. OLSON

The architecture of the Central Logic and Control computer (CLC) represents the first reduction to practice of large scale multiprocessing in a computing system. Multiprocessing was chosen because of its potential for high performance, while providing an architecture capable of high availability. The computer is significantly larger than previous multiprocessors which have used multiplicity numbers of the order of two or three. A modular concept is employed in which a community of as many as ten processors and a number of input/output elements share a common, modular memory.

The multiple elements communicate via well-defined interfaces and are interconnected by a flexible switching network. Control of the switching network allows the computer elements to be partitioned into two independently operating computers. The primary partition is large enough to run the real-time software, while the secondary partition is used for system exercise or for maintenance testing. Partitioning of elements is controlled by software, and reconfiguration may be accomplished in less than a second. A means of system recovery is built into the design to automatically restore real-time operations whenever the primary partition malfunctions.

PROCESS DESIGN—THE STRUCTURING OF REAL-TIME SOFTWARE SYSTEMS—W. S. DOYLE and J. R. GIBBONS

The term *process* is used to describe a complete software system for the CLC computer. Applications processes, driver processes, and test processes have been developed. Each of these includes an operating system as well as the applications programs.

Process design can be thought of as system engineering of the software components that constitute a process. The goals of process design are to identify the components of the process, to design a control structure that optimizes its execution, and to map the system design requirements across the process components efficiently.

The product of process design is a definition of the components of the process and a definition of the control structure.

This talk will discuss the techniques that were utilized in the design of two applications processes.

AN OVERVIEW OF THE CLC OPERATING SYSTEM—J. P. HAGGERTY

This talk surveys the capabilities of the operating system for the Central Logic and Control computer (CLC). The operating system is principally oriented toward applications software execution, i.e., toward providing the environment required by the programs which control and communicate with special peripherals. Non-real-time operations such as disk pack maintenance must also be allowed. The CLC operating system is therefore divided into two parts, the Tactical Operating System (TOS) for the former function, and the Basic Operating System (BOS) for the latter.

This talk discusses TOS at length, since the concepts behind this program are applicable not only to this project, but also to other computer systems which contain more than one central processing unit or which must respond to inputs in real-time. Emphasis is placed on the unique aspects of the CLC Operating System.

PROCESS-SYSTEM TESTING AND THE SYSTEM EXERCISER—B. P. DONOHUE, III and J. F. McDONALD

To establish a viable system test program, the following questions had to be answered: How would the system be exercised through its range of operation? What system parameters should be measured; and, how will the measurements be made? What is the success criteria against which the measurements will be compared, and what is the acceptable level of variation?

To exercise a system as complex as this through its range of operation is, in itself, a complicated task. To solve this, a tool called the *System Exerciser* was developed. It has the capacity of simulating a high traffic environment and various system peripherals, so that the bulk of the system hardware and software can be tested.

This talk addresses the development of the System Exerciser. In addition, it discusses the utilization of the System Exerciser in performing system testing. The concepts of developing a step-by-step test program are discussed, with emphasis on early planning and the utilization of the incremental approach.

SUPPORT COMPUTERS AND SOFTWARE—R. R. CONNERS

This talk focuses on one aspect of system implementation—the development, control, usage and operating environment of software tools such as the compiler, assembler, binder, and similar programs which, in turn, support development of applications programs.

Emphasis is placed on lessons learned over ten years of developing support software and over five years of operating commercial support computers, rather than on cataloging the programs which were written. However, there will be an introduction to each of the major support programs, and to the function of each of the commercial support computers used on the project.

THE USE OF FORMAL PROFESSIONAL REVIEW IN THE DEVELOPMENT OF SOFTWARE—R. D. FREEMAN

This talk describes "flowchart review," a technique that was used in the programming of one of the large pieces of software for this project.

The process of flowchart review required each programmer to write a detailed flowchart before doing any coding, and to give a box-by-box explanation of his flowchart to a review committee consisting of about a half-dozen of his colleagues. This review committee, in as friendly a manner as possible, was expected to take an "I'm from Missouri and you have to prove it to me" attitude, and to aggressively question every assumption that the programmer made.

In addition to greatly improving the quality of the code, there was a return in terms of software development time saved as a result of the time spent in these review sessions.

PROJECT CONTROL AND SUPPORT—H. M. JACKSON, II

This talk describes some of the unique management problems encountered during the development of the system. Topics addressed range from the management approach for directing the more than 2,000 people to some of the planning techniques that have proved useful in predicting the needs during the life cycle of the project for manpower, calendar time, computer time, etc. The plan used for documenting the software is presented along with a discussion of configuration management. Change control is discussed with special attention to control of changes to object code referred to as patches.

A panel session—Mathematical software—Patterns for the future

SESSION CHAIRMAN—JOHN R. RICE

Purdue University

Panel Members

Charles Lawson—Jet Propulsion Laboratory
William J. Cody—Argonne National Laboratories
William Gear—University of Illinois
Hans J. Oser—National Bureau of Standards
Barry Boehm—TRW Systems

OVERVIEW

The panelists present patterns that they believe to be important in the future of mathematical software. These patterns include the increasing demand for high quality, for a philosophy of development and testing, for standardized libraries with portability and the impact of computer networks, parallelism and lower hardware costs. Economic considerations will become more important as will the creation of software as the principal objective of research.

A panel session—Security kernels

SESSION CHAIRMAN—STEVEN B. LIPNER
The Mitre Corporation

Panel Members

William A. Wulf, Carnegie-Mellon University
Roger R. Schell, Major, United States Air Force
Gerald J. Popek, University of California
Peter G. Neumann, SRI Computer Science Group
Clark Weissman, System Development Corporation
Theodore A. Linden, Department of Defense

PANEL OVERVIEW—STEVEN B. LIPNER

INTRODUCTION

Recent experience in computer security has illustrated the susceptibility of numerous operating systems to hostile penetration.[1] Successful penetrations have been directed at manufacturers' conventional operating systems as well as special "secure" versions that have been the subjects of exhaustive efforts to find and fix all potential security problems. While formal reports are understandably hard to come by, it appears that the effort required to "break" any operating system and obtain access to any information it stores (at any time and without detection) is in the range two to four man-months. In contrast, the effort expended in futile attempts to prevent such penetration may be as much as two orders of magnitude greater (several man-years or more).

The underlying problem that is reflected by the ease of penetrating most operating systems is that of completeness. If even one error in an operating system allows a user to write a program that subverts the operating system's access controls, hundreds of other errors may have been corrected to no avail. In such a system, there is no relative protection or degree of security, but simply the (faint) hope that would-be penetrators will overlook the remaining error(s). If a penetrator does exploit such an error, his probability of success is one; the operating system's level of security is zero. Experience with penetration tests of current operating systems has shown the prevalence of errors of the sort mentioned. Unfortunately, it is the nature of the problem of completeness that no amount of testing by penetration attempts can demonstrate the absence of such errors.

SECURITY KERNELS*

The security kernel approach represents an attempt to find an alternative to the futile and (apparently) never-ending cycle of conducting penetration tests and correcting errors. The basic characteristic of this approach is the provision of a small correct mechanism, the security kernel, that implements the basic protection environment for the computer system. The kernel manages the resources of the computer in such a way that no program can subvert the access controls and effect a penetration. A few programs outside the kernel may perform ancillary security-related functions (viz: checking passwords at login) but they are not involved in the basic function of controlling access to information; the bulk of the programs in a system controlled by a kernel can have no effect whatever on security.

The security kernel approach is a radical one in the sense that it requires a new and different mechanism at the very base of the computer system. The idea of merely making minor repairs to an existing operating system seems much less drastic (and costly) and correspondingly more attractive. Unfortunately the latter approach has been tried and found unworkable. For this reason, the kernel approach is being pursued by a number of organizations with an interest in developing secure computer systems.

BASIC KERNEL REQUIREMENTS

The requirements that a security kernel must meet fall into two different classes:

(1) A kernel must present a suitable functional environment to the users and programs that employ it; and
(2) It must be designed and implemented so as to provide complete, correct and effective control over the environment it presents.

The differences in the several security kernels being developed by the panelists reflect differences in their perceptions of the requirements in each class and of the ways of meeting those requirements. The functional environments provided by the

* The work reported herein was sponsored by the United States Air Force Electronic Systems Division under Contract Number F19628-73-C-0001, Project 572R.

kernels vary from "bare" virtual machines through segmented virtual memories to very rich capability and domain environments. Similarly a variety of design, implementation, and proof techniques characterize attempts to identify and satisfy requirements of the second class.

Considerable variation in approach to defining a functional environment is a natural consequence of variation in user requirements. However, the requirement for sound kernel design and implementation is a constant if the problem of completeness is to be solved. It is success in meeting this requirement that distinguishes a sound security kernel, for the richest functional environment is of no value if the kernel that implements it can be penetrated. Accordingly, much of the panel's discussion will address the ways by which kernel developers can assure themselves and others that the problem of completeness has been solved.

KERNEL DESIGN AND PROOF

A number of issues in the areas of design, correctness, and proof bear on the success or failure of a kernel in solving the problem of completeness. Among these are:

—Interaction of function and correctness: Which sets of functional requirements facilitate the development of a correct kernel?
—Specification techniques: How does a designer specify a kernel's function? What characteristics of the functional description must be proven?
—Proof of design: How does the designer specify the mechanisms that implement a kernel? How can he convince himself that those mechanisms implement precisely the kernel he requires?
—Hardware-software tradeoff: The paragraphs above have left vague the question of whether a kernel is hardware or software. What hardware base is in fact required for a sound kernel, and how does it interact with kernel software?

SUMMARY

It is clear that this panel will not develop "the answers" to the questions above. However it is equally apparent that, in the coming years, a number of organizations will complete security kernels and describe them in the literature. This panel session should give its audience (and its panelists) ideas of what to expect of the security kernels to come and of how to assess them when they do become available.

REFERENCE

1. Branstad, D. K., "Privacy and Protection in Operating Systems" *Computer*, Volume 6, Number 1, January 1973, pp. 43-46.

HYDRA—A KERNEL PROTECTION SYSTEM*—WILLIAM A. WULF

The Hydra system is the 'kernel' base for a collection of operating systems designed to exploit and explore the potential inherent in a multiprocessor computer system. Since the field of parallel processing in general, and multiprocessing in particular, is not current art, the design of Hydra has a dual goal imposed upon it: (1) to provide, as any operating system must, an environment for effective utilization of the hardware resources, and (2) to facilitate the construction of such environments. In the latter case the goal is to provide a meta-environment which can serve as the host for exploration of the space of user-visable operating environments.

The particular hardware on which Hydra has been implemented is C.mmp,[1] a multiprocessor constructed at Carnegie-Mellon University. C.mmp permits the connection of (up to) 16 processors to 32 million bytes of shared primary memory through a cross-bar switch. The processors are any of the various models of PDP-11 minicomputers. Each processor is actually an independent computer system with a small amount of private memory, secondary memories, i/o devices, etc. Processors may interrupt each other at any of four priority levels; a central clock serves both for unique-name generation (see below) and broadcasts a central time base to all processors. Relocation hardware on each processor provides mapping of virtual addresses on that processor to physical address in shared primary memory.

The decision to use a 'kernel' approach, that is to provide only a set of mechanisms from which operating system facilities may be built, arose from two considerations: first, the conviction that only by this approach would it be possible to build a 'correct' system, and second, a desire to avoid predisposing the users of C.mmp to any particular mode of use through the idiosyncratic nature of 'the system'. (We want to learn how to effectively utilize a multiprocessor, *not* a multiprocessor plus particular operating system.) Thus our goal is to create an environment in which users can and will create their own operating environments—including scheduling and paging policies, file systems, etc.—and to support the simultaneous execution of an arbitrary number of such user-defined environments.

Given the general decision to adopt the 'kernel system' approach, the question remains as to what belongs in a kernel, and, perhaps more importantly, what does not. If a kernel is to provide facilities for building an operating system, and we wish to know what these facilities should be, then it is relevant to ask what an operating system *is* or *does*. Two views are commonly held: (1) an operating system defines a 'virtual machine' by providing facilities, or resources, which are more convenient than those provided by the 'bare' hardware, and (2) an operating system allocates (hardware) resources in such a way as to most effectively utilize them.

* This research is supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

Of course these views are, respectively, the bird's-eye and worm's-eye views of what is a single entity wiih multiple goals. Nevertheless, the important observation for our purposes is the emphasis placed, in both views, on the central role of *resources*—both physical and virtual.

The mechanisms provided by the Hydra kernel are all intended to support the abstracted notion of a resource (incarnations of a resource are called *objects*). These mechanisms provide for the creation and representation of new *types* of resources, as well as operations defined on them, protected access to instances of one or more resources within controlled execution domains, and controlled passing of both control and resources between execution domains. The key aspects of these facilities are the generalized notion of resource, the definition of an execution domain, and the protection mechanism which allows or prevents access to resources within a domain.

The mechanism used by HYDRA to achieve the various goals outlined above is an extension of the notion of a *capability* based protection system.[2,3] As with other capability systems:

(1) HYDRA supports the abstraction of a resource called an *object*. There may be several types of objects, typical examples of object types are pages, files, processes, and directories.
(2) For each type of object there are a finite number of operations, or *accesses*, defined on that object type. For example, operations such as 'open', 'close', 'read', 'write', and 'append' might be defined with respect to file-type objects.
(3) Objects are named by *capabilities*—which are protected by the system in the sense that a program may move them around, but may not create them or alter them in arbitrary ways. In addition to naming an object, a capability contains a specification of the accesses allowed to that object; that is, the capability determines the operations which may be performed on the object which it names. Hence, possession of a capability is *prima facia* evidence of the right to access an object in certain ways.

HYDRA extends the basic capability concept to allow any user to define new types of objects and operations on the new type. Thus the user may extend the basic resource types of "the system" as he sees it while not only retaining the protection applicable to previously defined resources, but extending the protection to the new resource. The user may wish to extend the system to provide a new facility (e.g., a different file organization), to alter management policies for a class of resources (e.g., a better disk request queueing algorithm), or to enforce a different security policy with respect to a resource. The extension of the capability model of protection involves several changes to the model:

(1) Objects may contain capabilities as well as data. (In the usual capability-based systems capabilities may only be kept in a special C-list associated with processes.) This permits existing objects to be used in the representation of new objects.
(2) The *operations*, or *accesses*, are defined by a special object type called a *procedure*. Since *procedures* are themselves objects, they may be manipulated, and protected, by the same mechanisms applicable to other objects.
(3) Protection is checked, as in other capability systems, when an operation is applied to an object; that is, when a *capability* is passed as a parameter to a *procedure*. In addition to this protection check, however, *rights amplification* may occur at this protection domain boundary (between caller and called). The called procedure may inherit *more* rights to the object than the caller possessed. This, in effect, allows the creator of a new object type to manipulate the representation of the objects of that type while preventing the owner of an instance of the type from doing so.

The HYDRA kernel is operational and is currently being used to explore the ease with which extensions of both function and security policy may be made in practice. To date our experience indicates that the HYDRA mechanisms can, indeed, be used to easily extend the environment of a user's program.

REFERENCES

1. Wulf, W. A. and C. G. Bell, "C.mmp—A Multi-Mini-Processor," *Proc. AFIPS FJCC 1972*, pp. 765-777.
2. Dennis, J. B. and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *CACM* 9, 3, March 1966, pp. 143-155.
3. Lampson, B. W., "Dynamic Protection Structures," *Proc. AFIPS Conf.*, 35, FJCC 1969.

EFFECTIVENESS—THE REASON FOR A SECURITY KERNEL—ROGER R. SCHELL

INTRODUCTION

From the viewpoint of protection, the primary technological deficiency of today's computer systems is the pervasive lack of effective internal security controls.[1] There is a sizeable collection of nominal access controls for information entrusted to computer systems; yet these often attractive protection features seem impotent when faced with a deliberate effort to thwart them. The military, in particular, needs dependable controls, since they have no legal redress for information compromised by an enemy. A suitable "security kernel" is one of the few available techniques for providing effective security.

Security kernel connotes a small, basic set of controls. However, a kernel needs more than just modular design, well-structured specifications, or similar useful (but not sufficient) techniques. We advocate three elements of kernel development that provide a basis for meaningful protection: (1) precise security definition; (2) fundamental design; (3) certifiable implementation.

PRECISE SECURITY DEFINITION

The first step in kernel development is clearly establishing what it means for the computer system to be "secure". The abstraction of a "reference monitor"[3] controlling the access of active *subjects* to *objects* is used to model the security characteristics of the entire system—for the military this models the access of people to information. For the computer component of the system there are corresponding representations. With this model, *all* operations in the system can be considered in terms of just two security (equivalence) classes: (1) the *Reference* function provides for subjects to access objects based on a set of access authorizations; (2) The *Authorize* function provides for subjects to change the access authorizations. The specific formulation of these two functions precisely determines the protection features of the system.

System features may range from total isolation (as between distinct virtual machines) to refined control of information sharing. The military controls access based on individual "need-to-know" and a set of (partially ordered) national security "classifications". We have modeled[2] secure (against compromise) military systems by a *Reference* function that allows a user access if and only if his classification (called "clearance") is greater than or equal to the information classification and he has a need-to-know. Similarly, the *Authorize* function permits access authorization only to sufficiently cleared users. This model is secure even against "trojan horse" attempts to violate classification. A precise model is basic, with or without a security kernel; however, we cannot today prove an entire system correct, but it is sufficient to concentrate on the kernel.

FUNDAMENTAL DESIGN

The kernel design for the reference monitor must faithfully reflect the formal model. This faithfulness is characterized by three design principles: the kernel is (1) *complete* in mediating all references; (2) *isolated* from tampering; (3) *simple* enough that its correctness can be established. Our basic approach is to partition the security controls into a distinct portion of the system. The design is fundamental in the sense that it is independent of computational semantics. That is, the kernel is demonstrably sufficient for system security (as precisely defined) for *all possible* computations, including any penetration attempts.

The effectiveness of this approach is in marked contrast to the usual approach to security controls, not only in degree but also in basic concept. Nominal controls represent essentially a game of wits between the designers making all the checks they can think of and a penetrator looking for one oversight. On the other hand the viability of the postulated approach has been demonstrated in our recently completed prototype design.[5] The precise definition of "secure" and a fundamental kernel design approach provide the basis for establishing *a priori* (viz., "certifying") that a system is secure.

CERTIFIABLE IMPLEMENTATION

The essential characteristic of a certifiable system is the specific design criteria which, if met, insures the algorithmic security of the system. This certification is distinguished from efforts to directly test system security (e.g., by penetration teams) and from *ad hoc* attempts at improved implementations for "more secure" systems. Certification must establish the correspondence between the fundamental design and the implementation. One approach[4] is a direct mathematical proof—by hard work and tenacity. Alternatively, a structured implementation (after the manner of Dijkstra) may be developed by successive decomposition of the *Reference* and *Authorize* functions. Hardware support for per segment access control and multiple protection environments (e.g., as in Multics) is also a major implementation aid.

Since this kernel development approach is implicitly based on a deterministic, asynchronous view of computation, a final comment is in order: the problems of hardware component failures and the use of external time to "send Morse code" must be independently considered.

CONCLUSIONS

A properly constructed security kernel can provide effective internal access controls for modern computer systems. A sufficient theory exists, design feasibility has been demonstrated, and adequate hardware/software architectures for implementation are available. The industry must now choose between a sound (security kernel) basis for truly effective security or continued costly and unsuccessful attempts to correct the weaknesses of contemporary systems.

REFERENCES

1. Anderson, J. P., *Computer Security Technology Planning Study*, ESD-TR-73-51, October 1972.
2. Bell, D. E., *Secure Computer Systems: Mathematical Foundations*, ESD-TR-73-278.
3. Lipner, S. B., *Computer Security Research and Development Requirements*, MTP-142, MITRE Corporation, February 1973.
4. Price, W. R., *Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems*, Ph.D. Thesis, Carnegie-Mellon University, June 1973.
5. Schiller, W. L., *Design of a Security Kernel for the PDP-11/45*, ESD-TR-73-294, June 1973.

A PRINCIPLE OF KERNEL DESIGN—GERALD J. POPEK

One of the results of the recognition of the need for reliable control of access to information has been attention to the way that software in multiuser systems is structured. The concept of a security kernel as the lowest level of an operating system, containing just that code relevant to security, has grown out of that attention.

The value of a security kernel, as well as a discussion of the idea itself, is presented in companion notes here, and by the author elsewhere in this volume. Here we attempt to generalize one of the notions behind the value of a security kernel by describing a principle of which the kernel is an application. We name it the principle of *"least common mechanism."**

A common mechanism shall be one which, if it were to operate improperly, could lead to a security flaw. For many of today's systems, the entire operating system is a common mechanism, for it is run in supervisor state with universal privileges, while applications programs are not. The principle demands that a system be designed to minimize the amount of such common mechanism, and we argue that if the principle is followed the security of the system will generally be enhanced. An example may provide some clarification.

Most contemporary operating systems provide an I/O interface for user programs that is considerably improved over that of the bare machine. Consequently, however, a significant amount of code is often devoted to translating user requests into a form directly processible by the hardware. The code is frequently complex and a rich source of security flaws. In most systems today, that translation is a common mechanism. It is executed on behalf of many users, in supervisor state, with high privileges.

However, the above structure is not intrinsically necessary. The only portion of the code which must be a common mechanism is that which, at the very last moment possible, checks the hardware I/O commands and actually empowers the action. In that way, security is not dependent on translation. Execution errors during translation are not worrisome since they occur in an impotent environment without special privileges, and the common checking code catches any illegal I/O commands that may have been issued. The amount of common mechanism has been considerably reduced.

Note that the translation code can be reentrant, and still shared by many users. This sharing does *not* make the code a common mechanism. True, errors in that code may cause a user program to malfunction; in fact many user programs may fail because of the same error. However, no security breech occurs. Each user is still limited only to the information which is rightfully his. The conditions here are similar, for example, to bugs in a system supplied compiler, which may cause the failure of many user programs but without security implications.

*This name was suggested by Mike Schroeder during a conversation in which this author was explaining the security advantages of virtual machine designs.

As another way that the amount of common mechanism can be reduced, suppose a system contains extensive facilities to support dynamic interprocess parameter passing. Suppose further that users rarely if ever employ them to communicate between jobs with differing access privileges, although the facilities are often used within a given job. These facilities should not be run in privileged mode, but rather placed in a less powerful environment. For example, a virtual machine monitor or hypervisor removes many service features of a system (in fact, entire operating systems) from the position of common mechanism.

It should be pointed out that, in both of the above examples, it usually would be necessary to restructure the guts of the operating system innards. The code which performs I/O translation and checking is frequently not easily separated.

The principle of least common mechanism can be a considerable aid in improving the security of a system and helps determine what code a security kernel should contain. Exactly the minimum common mechanisms should be included. However, a particular architecture has not been specified. As a case in point, it is possible that a system should have more than one kernel while still agreeing with the principle. For example, let the basic kernel of a general purpose operating system provide for supporting and isolating simple processes. The only primitive sharing facility included might be the ability to arrange shared read-write segments. Then, a file system, with its own kernel, could be built in one process, and whenever a user wished to act on a file, he would communicate with the file system through a shared segment. The original kernel's task of process isolation and the responsibilities of the file system's kernel have been separated, decreasing complexity, and the upper levels of software each needs to support have also been simplified.

The effect of the definition of the common mechanism depends on what is meant by a security flaw. For example, one might mean only that users are to be prevented from obtaining information that belongs to others. This is the usual viewpoint current today, tacitly assumed here. It includes for example obtaining illegal access to another user's files. However, an alternate interpretation of security might demand that there be no way by which one user can cause a defective scheduler not to run another user. One's definition of security can have a significant impact on what portions of a system are considered common mechanism. In the above, the scheduling discipline must be included.

It has been suggested[1] that one way to judge the security design of a system is to evaluate it with regard to the principle of least privilege: is it possible to limit users' access to the least amount of information and power necessary to perform the desired task.[4] For example, most traditional operating systems violate this principle, since when the user calls upon the supervisor for a service, he cannot limit the supervisor only to the relevant information. Protection that is efficient only for large files, or available only for entire directories, implies that a user often must be given access to more than is necessary.

The concept of least common mechanism should also be used as a judgment criterion, for the less that is involved in maintaining the security of the system, the less that must be certified. In fact, for some multiuser systems, those parts upon which the security of the system depends may be made small enough and straightforward enough in their structure to admit of a formal verification of their security properties, or perhaps even become candidates for firmware implementation.

The first two examples in this note could have been justified by least privilege rather than least common mechanism, although those changes in structure are unlikely to have been motivated that way. Nevertheless, there are differences between these two principles. Whether the I/O checking code mentioned earlier has access to the contents of various devices or sections of memory is unrelated to its being a common mechanism. Under certain conditions that code need not even be run in a highly privileged mode.

However, these principles of least common mechanism and privilege should be viewed together, for strict adherence to one can cause difficulties with the other. When to trade extra mechanism for occasional violation of least privilege, and vice versa, may not be clear. The ability to control access to individual data elements may support least privilege, but if it is rarely useful, then the extra mechanism needed to support that ability may not be justified.

The principle of least common mechanism is the underlying reason why kernel designs can in general lead to more secure systems.

BIBLIOGRAPHY

1. Jones, A. K., *Protection in Programmed Systems*, Ph.D. Thesis, Carnegie-Mellon University, 1973.
2. Linden, T. A., "A Summary of Progress Toward Proving Program Correctness," *AFIPS Conference Proceedings*, 1972 Fall Joint Computer Conference, pp. 201-211.
3. Popek, G. J. and Kline, "Verifiable Secure Operating System Software," *AFIPS Conference Proceedings*, 1974 National Computer Conference.
4. Saltzer, J. H., "Protection and the Control of Information Sharing in Multics," to be published in *Communications of the ACM*.

ON THE DESIGN AND VERIFICATION OF A SECURE OPERATING SYSTEM—PETER G. NEUMANN

Claims regarding the security of an operating system are meaningful only if the security can be verified, i.e., the operating system can be shown to satisfy a set of explicit security requirements. Even then, these requirements must themselves be complete enough to preclude unanticipated security violations. Various factors affecting the feasibility of obtaining such verification are discussed below. For example, extensive use of structure and compartmentalization can greatly facilitate design, implementation, and proof of security.

In order to achieve the objectives of demonstrable security, it is very desirable to have the following tools and techniques:

(a) A formal design methodology encouraging a highly structured design, and providing design descriptions, or *specifications*, at each of various levels of the design.
(b) A formal assertion language in which to state precisely the desired security requirements.
(c) Techniques for proving that the design and the implementation satisfy the formally stated security assertions. These techniques should take advantage of any structure in the design and implementation. They may be mostly manual or computer assisted, but there is growing hope for the further development of realistic semi-automatic proof techniques.

A very attractive approach to proving the security behavior of a complex operating system is to partition the design so that all security-sensitive functions are clearly partitioned, e.g., those functions controlling sharing and isolation of protectable objects. (We refer to the boundary of such a partition as the "security perimeter." It generally contains significantly more than the "security kernel" referred to elsewhere in this session.) Proof of security then consists of proofs regarding the interior of the security perimeter, plus proofs of successful isolation, i.e., that portions of the system outside of the security perimeter cannot compromise the security. The interior of the security perimeter may itself be highly structured (although not necessarily hierarchical).

A small team of SRI people (Karl Levitt, Larry Robinson, John Wensley and myself) is currently undertaking to design a secure general-purpose operating system (with main emphasis on the design within the security perimeter) and to prove that this design satisfies the security requirements. We are developing a formal design methodology based on Parnas' technique of module specifications,[1-3] and extending it to make it suitable for a large system consisting of many modules. Extensions include the representation of intermodule connectivity and the well-defined handling of exception conditions. The formal assertion language is closely related to the specification language, although somewhat more precise. The proof techniques are evolving out of our experience in the use of a range of current techniques for proofs of correctness of smaller programs.[4-5] These proof techniques are highly structured, and are appropriate to our structured design. We expect that the large size of the entire operating system will not be a deterrent to provability, since the design is being structured into reasonably clean interconnections of reasonably well-structured and modest-size modules. As a result, the complexity of proof can be kept approximately linear with module size and with interconnection complexity. We feel that this approach holds great promise. In particular, we believe that these techniques are adequate for our present purposes, at least with respect to manual proof.

The design is using a capability-based design philosophy [e.g., Reference 6] with extendible object types. Although we are initially attempting only to prove a detailed design (that is, a "partially implemented" design), the proof techniques also apply to the proof of an ultimate implementation (in terms of executable programs) as the lowest levels of design. In this sense, proof of an implementation is a multistage process iteratively involving proofs of various stages of design (or implementation) specificity. Thus verification of the security of the entire operating system is a relatively straightforward effort; given the proof of the detailed design. Furthermore, it is in general sufficient to reprove selectively following local modifications.

We have one unusual consideration, namely that we are not constrained by inadequacies of any existing architecture. This is for the most part a great advantage. However, it is sometimes necessary to resist temptations toward conceiving infeasible architectures. A final comment is in order on the scope of our work. We feel very strongly that hardware error detection, crash recovery, and fault-tolerance in general are intimately related to security, and thus we are including such concepts in our design considerations. We hope that our results will have a general and wide applicability.

REFERENCES

1. Parnas, D. L., "A Technique for the Specification of Software Modules with Examples," *CACM* Vol. 15, pp. 330-336, December 1972.
2. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *CACM* Vol. 15, pp. 1053-1058, December 1972.
3. Parnas, D. L. and D. Sieworek, *Use of the Concept of Transparency in the Design of Hierarchically Structured Systems*, Carnegie-Mellon, 27, July 1972.
4. Elspas, B., K. N. Levitt, R. J. Waldinger and A. Waksman, "An Assessment of Techniques for Proving Program Correctness," *Computing Surveys* Vol. 4, No. 2, 1972.
5. Waldinger, R. and K. Levitt, "Reasoning about Programs," *ACM Proc. Symp. on Principles of Programming Languages*, pp. 169-182, October 1973.
6. Lampson, B. W., "Dynamic Protection Structures," *AFIPS Conf. Proceedings, 1972 Fall Joint Computer Conference*, Vol. 41, pp. 33-47, 1972.

KERNELS FOR VARYING SECURITY REQUIREMENTS—T. A. LINDEN

In discussing security kernels it is useful to begin by focusing on the word "security" rather than on the word "kernel". Security can mean many different things. Fruitless arguments may be avoided if we take the time to define the range of possible meanings for security and the way these meanings interact with the kernel idea.

To choose a specific meaning for the words "secure system", I propose that all three of the following questions must be answered:

(1) What class of threats is the system secure against?
(2) What type of evidence for security is expected?
(3) What types of sharing will be required in the system?

With respect to the first question, security kernels attempt to deal with a reasonably well-understood class of threats. They provide access controls in the hardware and resident software which can prevent simultaneous users of the system from interacting in undesirable ways. Many other types of threats, such as those associated with the physical integrity of the computer site, are not addressed by the kernel idea. However, a security kernel can make it easier to handle some additional threats. For example, since a security kernel isolates the security-relevant software in a well-defined part of the whole operating system, it becomes easier to guarantee that it is loaded correctly and to control modifications to it.

More significant insight into the kernel idea can be gained by turning our attention to the second question—the level of evidence available to support claims of security. When discussing software access controls, I suggest that the evidence for security is usually based on one of the five following approaches:

(1) Study of the manufacturer's manuals and brochures.
(2) Personal confidence in the system's designers and implementers.
(3) Analysis of the system *design*.
(4) Testing and penetration efforts.
(5) Analysis and proof of assertions about the detailed implementation.

It is indicative of the present state of the art that it is hard to stop smiling until one approaches the end of this list. Even extensive testing and penetration studies provide little evidence that a system is secure, although they frequently prove the opposite.

I believe that the essential point to a security kernel is that it should make it feasible to provide solid objective evidence for the security of the access controls implemented in software. It is not enough that the access controls be designed just to meet specified security threats; they must also be designed so that the integrity of the implementation will be demonstrable. Given our present ability to demonstrate properties of programs, this puts severe limitations on the size and organization of both the programs that implement the access controls and all the software and hardware on which they depend.

The types of sharing that will be required from a secure system account for most of the differences between various approaches to security kernels. The first aspect of this problem is the sharing of physical resources such as the CPU, memory, peripherals, etc. The problem of designing a small, well-structured security kernel clearly becomes more difficult as more sharing of physical resources is required. Most current security problems arise from the need to share physical resources; however, current trends in hardware costs indicate that the most important long range security problems may arise from the need for other types of sharing.

In the future we can expect a continually increasing need for flexible sharing of programs and data. This may entail much more than just a file system that allows restricted distribution of the programs and data. For example, if a program written by one person is to be used by someone else, then we might want the security kernel to guarantee that the program cannot steal or alter all of the information in the file directory of the person who used the program

Finally, there is the problem of sharing or decentralizing of control over who can access what. Centralized security control is adequate in many situations—in fact is often necessary, especially for the military. However, there are situations where decentralization of the security controls would be very desirable.

In discussing security kernels, it is important to carefully consider the amount of sharing which will be needed. Complex forms of sharing will certainly make it more difficult to implement a small, provable security kernel. On the other hand, if a system does not allow for enough sharing, then manual procedures will usually develop which effectively bypass or negate the carefully designed automatic security controls.

A panel session—Snobol languages

SESSION CHAIRMAN—RALPH E. GRISWOLD

University of Arizona

Panel Members

James F. Gimpel—Bell Laboratories
Robert B. K. Dewar—Illinois Institute of Technology
Paul J. Santos—Bell Laboratories
David R. Hanson—University of Arizona

OVERVIEW

SNOBOL has developed from simple string processing to a sophisticated general purpose programming language. This session will consist of brief presentations and discussions by panelists who have played major roles in the design, implementation, and application of the SNOBOL languages. The panel will be available for questions from the audience.

A panel session—Program debugging

SESSION CHAIRMAN—HELENE E. KULSRUD

Institute for Defense Analyses

Panel Members

Robert M. Blazer—USC/Information Sciences Institute
Hermann H. Goldstine—IBM and The Institute for Advanced Study
Ralph Grishman—Courant Institute, NYU
M. D. McIlroy—Bell Telephone Laboratories
P. E. Hagerty—University of Maryland
Moderator
Helene E. Kulsrud—Institute for Defense Analyses

OVERVIEW

The computing community is now concerned with the cost in time and money consumed by the debugging process. In spite of advances, such as proving program correctness, use of higher level languages and new methods for structuring programs, the problem remains. The panelists will consider many phases of the debugging activity as it exists today, giving their views on current and proposed tools and techniques.

A panel session—Computer output to microfilm

SESSION CHAIRMAN—JOHN R. RIDGEWAY

Combustion Engineering, Inc.

Panel Members

James N. Hollister—Eastman Kodak Company
Jon Warms—American Telephone & Telegraph
Bryan Wood—Gould. Inc.
Tom Glacken—Computer Microfilm International Corp.

OVERVIEW

This session assumes no prior knowledge of COM. We will review a history of COM, what it is, where it is going and then, related COM to Information Management. Micro film technology can be used in a real-time MIS environment as a replacement for computer devices; as an extension of the Computer's capabilities; and as a low cost back-up system.

A panel session—The CODASYL and GUIDE/SHARE proposals on data base management systems

SESSION CHAIRMAN—LARRY SIMONETTE

Peat, Marwick, Mitchell

Panel Members

Barbara Fossum—Sperry-UNIVAC
Chuck Mairet—Deere & Company
C. W. Bachman—Honeywell Information Systems Division
Mike O'Connell—Digital Equipment Corporation
Roger Holliday—IBM Corporation
Don Jardine—Queens University

OVERVIEW

Six panelists will discuss the subject of Data Base Management (DBM) as it relates to CODASYL's Data Base Language Task Group report and Data Definition Language Committee Report and the GUIDE/SHARE published requirements for a DBM and other published GUIDE documents deemed pertinent. Four key issues that best deal with the philosophy, concepts and differences of the CODASYL and GUIDE/SHARE work to date will be addressed.

A panel session—Report on the IBM data security study

SESSION CHAIRMAN—LEE DANNER

IBM Corporation

Panel Members

Robert T. Caravella—State of Illinois—MID
Robert H. Scott—MIT—Information Processing Services
Gerald E. Short—Systems Group of TRW, Inc.

OVERVIEW

The IBM Data Security Study Project has essentially been completed. This session will summarize IBM's efforts to the project and reports by three non-IBM study sites. The discussion will cover data security from the viewpoints of the user, the system designer, and the administrative and operations managers.

A panel session—Audit considerations of data bases

SESSION CHAIRMAN—RICHARD A. NERAD

Arthur Anderson & Company

Panel Members

Robert Manion—Author Anderson & Company
Gresham T. Brebach—Arthur Anderson & Company
James Muenz—Kraftco Corporation

OVERVIEW

Now, at the onset of a new era of data processing centered around the data base concept, it is important for the business-oriented systems man to be aware of the impact that this concept has on the company management responsible for the business operations. This panel will present a series of topics related to the corners of key corporate responsibilities: the executive management, the financial management and the auditor.

A panel session—Research in data security—Policies and projects

SESSION CHAIRMAN—ROBERT F. MATHIS

The Ohio State University

Panel Members

Stephen R. Crocker—Advanced Research Projects Agency
Marvin Denicoff—Office of Naval Research
Victor Mitchell—U.S. Army Material Command
Frederick W. Weingarten—National Science Foundation
Edward Feustel—Rice University
Lance J. Hoffman—University of California—Berkeley
David Hsiao—The Ohio State University
Rein Turn—The Rand Corporation
William Wulf—Carnegie-Mellon University

SCOPE AND SIGNIFICANCE—ABSTRACT

This session will begin with presentations and a panel discussion by representatives of various governmental agencies involved with research or funding of research in data security. They will present their views on the policies and goals of federally funded research in the area of data security and how this relates to privately funded research and development in this and related areas. They will also have the opportunity to point out new directions which should be pursued. There will be discussion on both the technological and social impact of such funding.

The session will also include reports from some significant research projects in the data security area. The research projects will be described in terms of original scope and purpose, results obtained, investigations underway, suggestions for future research, and publications. These research reports should provide the computing community at large a good view of current research, support policy, and future directions in the data security area.

This session will differ substantially from, but be complementary to, the sessions on user oriented development projects in data security which were presented at the last National Computer Conference. It should provide a good forum for reporting the current status of many research projects in the area of data security and serve as a means of encouraging communication between researchers and users interested in this area.

RESEARCH ON COMPUTER SECURITY COSTS AT THE UNIVERSITY OF CALIFORNIA, BERKELEY—LANCE J. HOFFMAN

Research into costs associated with computer security techniques is currently underway at the University of California, Berkeley. Several types of costs are being investigated.

ENCIPHERMENT/DECIPHERMENT

Costs of encipherment and decipherment have finally been determined in replicable experiments on Berkeley's CDC 6400 and the Stanford IBM 360/67. Using the results of these experiments, encipherment time coefficients were calculated for four different encipherment methods: one-word key multiword key, double key, and pseudo-random key. The encipherment time coefficients, which measure the relative speed of a given encipherment method on a given machine are shown in Table I, and the interested reader is referred to Reference 1 for more detail.

Preliminary worst-case overhead results have also been measured for a method of altering operating systems to allow dynamic data-dependent decisions at execution time without requiring either the binding of security parameters at compile-time, or the great increase in overhead necessitated by run-time decisions. Relatively simple additions to the functions of input-output routines may result in the dynamic capabilities we want at little additional cost. By interposing a security module between user programs and system input/output routines (see Figure 1), we have been

TABLE I—CDC 6400 CPU Time Used for Encryption

Method	Encipherment Time Coefficient Assembly Language	Encipherment Time Coefficient FORTRAN
Null Transformation	1.000	1.00
One Word Key	1.004	2.68
Long Key	1.73	4.03
Double Key	2.64	6.60
Pseudo-Random Key	4.21	9.96

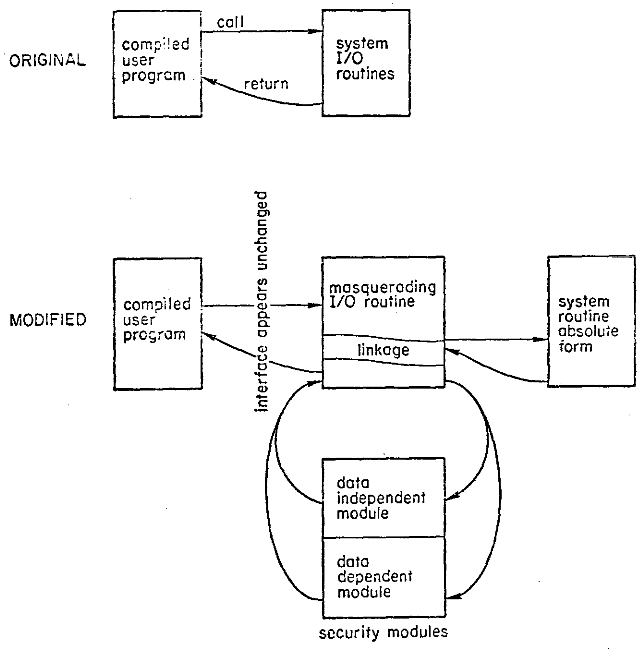


Figure 1—Modified system I/O to implement data security modules

able to achieve dynamic security checking with speed approaching that of data-independent decisions in a manner applicable to most operating systems today.[2] We plan to extend and improve our preliminary CDC 6400 results on the IBM 360/67 and hope that additional CPU overhead can be cut to well below 10%.

We initially planned to design several application-oriented languages which would incorporate data-dependent and data-independent security features and we still plan to design some application-oriented languages with very general security features after the work of the previous paragraph is completed. It is our hope that some of these will be able to screen out most unauthorized data accesses at compile-time and still support dynamic data access decisions at run-time.

We will also continue our work with procedure-based access control mechanisms[3] and hope to eventually implement some of these in microcode or hardware and thus eliminate a great deal of CPU and memory overhead which is caused by security considerations. We also hope to develop a theoretical framework for procedure-based access control analagous to and compatible with access matrix models and other models.[4]

The work described above is all being done under NSF Grant GJ-36475. It is separate and distinct from another project on computer security at Berkeley, the PRIME Project.[5]

REFERENCES

1. Friedman, T. D. and L. J. Hoffman, *Execution Time Requirements for Encipherment Programs*, University of California Electronics Research Laboratory Memo ERL-M378, June 1, 1973. To appear, *Communications of the ACM.*
2. Woodward, F. G. and L. J. Hoffman, *Worst-case Costs for Dynamic Data Element Security Decisions*, University of California Electronics Research Laboratory Memo ERL-M413, September 26, 1973.
3. Hoffman, L. J., "The Formulary Model for Access Control," *Proc. AFIPS* 1971 FJCC, Vol. 39, AFIPS Press, pp. 587–601.
4. Bell, D. E. and L. J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, ESD-TR-73-278, USAF Electronic Systems Division, L. G. Hanscom Field, Bedford, Mass.
5. Baskin, H. B., B. R. Borgerson, and R. Roberts, "PRIME—A Modular Architecture for Terminal-Oriented Systems," *Proc. AFIPS* 1972 SJCC, pp. 431-437.

RESEARCH ON DATA SECURE SYSTEMS*—DAVID K. HSIAO, DOUGLAS S. KERR AND FRED A. STAHL**

The research at The Ohio State University on data security has evolved from some of the earlier work in logical access control mechanisms begun in 1965.[1] In order to provide an introduction for others to some of the work, we have published a summary entitled "Logical Access Control Mechanisms in Computer Systems."[2] Due to considerable advances in computer hardware and software protection mechanisms in recent years, an examination of the logical access control mechanisms with respect to their supporting hardware and software protection mechanisms is needed. To this end we have taken the first step by reviewing critically some of the significant work in the area of data security. The review is documented in a technical report entitled "An Annotated and Cross-Referenced Bibliography on Computer Security and Access Control in Computer Systems."[3]

Our progress in developing a general theory of data security and data secure systems has been good. Experimentations will occupy our attentions for the months to come. Major research and experimentation efforts are discussed below.

ON DEVELOPING A THEORY OF DATA SECURITY

A multi-level model for data secure systems has been proposed.[4] In this model the relevant issues in data security, such as integrity, privacy protection and controlled information sharing, can be studied, on the one hand; and the conventional procedures such as identification, authentication, authorization, and compartmentalization can be characterized, on the other hand. Furthermore, the model allows different problems in data security to be considered at a level of abstraction appropriate to the specific issue and procedure under study. The highest level is *conceptual*. In it, "patterns of protection" (intuitively, the ways the users may access the data) can be defined in formal and unambiguous

The work reported herein is supported by the Office of Naval Research under Contract N00014-67-A-0232-002.

** The Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210.

ways. The intermediate level of the model is *structural.* Here, the primitives to be utilized in the realization of the patterns of protection defined in the higher level will be specified. The most important feature of this level is that the critical functions of an access control mechanism are no longer carried out by complex, and thus potentially unreliable programs, but are inherent in the basic structure of the system by the utilization of deadlocks. When a user attempts an unpermitted access, he deadlocks with a "pseudo-user" and cannot proceed. Thus, the demonstration of system correctness involves the certification of a limited number of small, single-purpose modules and the verification of the correctness of the user/pseudo-user interaction. On the lowest level, a *system* to illustrate the utility and practicality of the model will be created. Overall, the research should suggest a modelling and design technique for a demonstrably complete and correct system for providing logical access control in a shared data base system.

Our present plan for this research consists of three studies. The first is to complete our abstract model of data secure systems and develop a general theory of data security as proposed in Reference 4. In particular, we emphasize the structural level of the model. It is hoped that this level of modelling can reveal the inner working of the access control mechanism based on the theory of deadlock. With a good understanding of its inner working, the access control mechanism can then be properly designed and implemented. The application of the theory of deadlock to access control is new. We believe that this is the first application. Traditionally, system designers attempt to avoid and circumvent the system deadlocks which tie up system resources and utilities. However, in our case, we deliberately tie up resources and utilities as a means to deadlock penetrators of the system. Obviously, these resources and utilities are logical resources such as files, records and fields and functional utilities such as data access and manipulations. Such deadlock is called *security deadlock.* One of the basic requirements is that no authorized use of and access to the data base will cause a security deadlock and any unauthorized use or access will cause an immediate security deadlock. This requirement will be met.

ON CONTEXT PROTECTION AND CONSISTENT CONTROL

Although the capability of the access control mechanisms to regulate field, record and file security, has been recognized as indispensible in advanced data secure systems, there is the need of more subtle protection and refined control which we shall call *context protection* and *consistent control.* Context protection enables the same data unit (field, record or file) to be protected differently in different contexts. For example, the same data field may be protected differently in different records. The difference may be determined by the manner in which the fields and records are being accessed. Consistent control is concerned with the problem that when new data units based on the old data units of the data base are created by the users, these data units must be protected consistently in the sense that their access requirements must be generated automatically and must conform with the access requirements of the old data units. Our study[5] has begun to show that both context protection and consistent control can be enforced by means of certain built-in relations among the data units involved. These relations under certain conditions can reveal any violation of context protection and consistent control. Here our first step is to identify those relations which are basic and primary to the contextual relations. It is hoped that by proposing these basic and primary relations, more elaborate contextual relations among data elements can be defined for protection reasons. Furthermore, a method of enforcing the protection can be facilitated by these basic and primary relations. Necessary and sufficient conditions under which a protection violation will occur due to contextual security constraints must be identified. With these conditions and the method, it will be possible to propose a data definition language for specifying data base protection requirements and to develop an access control mechanism for enforcing the requirements.

ON DATA SECURE COMPUTER ARCHITECTURE

In this research we are concentrating on computer architectural (hardware) requirements for the realization of data secure systems.[6] Hardware requirements will be investigated via hardware modification (such as microprogramming) of an existing computer system. By introducing a special data management instruction repertoire for access control mechanisms, the problem of high performance and reliability may be alleviated. It is the primary aim of this study to determine the desired computer hardware organization for the enhancement of data security. Systems requirements will be studied in view of computer configuration and architectural issues. To this end, we intend to build some special software/hardware interfaces interconnecting several different computer systems. Novel system architecture utilizing a high degree of processing parallelism and distributed functionality may provide new solutions to improved effectiveness in data security.

ON DESIGN AND IMPLEMENTATION TOOLS FOR DATA SECURE SYSTEMS

In order to demonstrate the feasibility and usefulness of the theoretical studies, an undertaking on system design and implementation of a demonstration system is desired. Furthermore, the system will be needed as a vehicle to experiment with new access control mechanisms as well as novel data base applications. As a design and implementation tool, a PL/1-like systems programming language, known as PL/X, has been developed.[8,9,10] We plan to use PL/X for the design and implementation of experimental data secure systems.

ON DISTRIBUTED SECURE DATA BASES IN COMPUTER NETWORKS

One of the most promising ways of utilizing network capabilities while providing efficient and reliable processing is the establishing of distributed data bases. However, distributed data bases pose unique problems and advantages toward the safeguarding of sensitive information. In our research[11] we are investigating the ways in which cryptographic techniques can be employed to provide the needed security inherent in transmitting information over common carrier lines. In particular, we are investigating techniques by which information in encrypted form may be processed without passing through the additional steps of decryption and subsequent reencryption.[12,13]

REFERENCES

1. Hsiao, D. K., "Access Control in An On-Line File System," *Working Papers of FILE68—International Seminar on File Organization* by I.F.I.P. (I.A.G.) Administrative Data Processing Group, Copenhagen, Denmark, Nov. 1968. The same paper is also included in a book entitled *File Organization—Selected Papers from FILE68, an I.A.G. Conference* by Studentlitteratur Ab, Lund, Sweden.
2. Hsiao, David K., "Logical Access Control Mechanisms in Computer Systems," *Proceedings of Conference on Secure Data Sharing*, ONR/NSRDC, Washington, D.C., July 1973.
3. Bergart, J. G., M. Denicoff, and D. K. Hsiao, *An Annotated and Cross-Referenced Bibliography on Computer Security and Access Control on Computer Systems*, Technical Report OSU-CISRC-TR-72-12, The Department of Computer and Information Science, The Ohio State University, November 1972.
4. McCauley, E. J., III, and D. K. Hsiao, *A Model for Data Secure Systems*, Technical Report OSU-CISRC-TR-73-8, The Department of Computer and Information Science, The Ohio State University, February 1974.
5. Nee, C. J., and D. K. Hsiao, *Theoretical Foundations for Context Protection and Consistent Control in Data Secure Systems*, Technical Report OSU-CISRC-TR-73-9, The Department of Computer and Information Science, The Ohio State University, February 1974.
6. Baum, R. I., and D. K. Hsiao, *A Data Secure Computer Architecture*, Technical Report OSU-CISRC-TR-73-10, The Department of Computer and Information Science, The Ohio State University, March 1974.
7. Hsiao, D. K., and T. Wyrick, *System Configuration Study on the Interconnection of the IBM 370/165, DEC System-10 and Micro 1600/21 Computer Systems*, Technical Report OSU-CISRC-TR-73-7, The Department of Computer and Information Science and the Instruction and Research Computer Center, The Ohio State University, October 1973.
8. Hsiao, D. K., et al., *The PL/X Compiler-Subsystem Writers' Manual*, The Instruction and Research Computer Center and The Department of Computer and Information Science, The Ohio State University, August 1973.
9. Hsiao, D. K., et al., *The PL/X Programming Language Reference Manual*, The Instruction and Research Computer Center and The Department of Computer and Information Science, The Ohio State University, August 1973.
10. Hsiao, D. K., et al., *The PL/X Compiler Program Statements*, Technical Report OSU-CISRC-TR-73-11, issued jointly by the IRCC and The Department of Computer and Information Science, The Ohio State University, December 1973.
11. Cohen, E. I., D. K. Hsiao, and F. A. Stahl, *Distributed Secure Data Bases in Computer Networks*, Technical Report OSU-CISRC-TR-73-11, The Department of Computer and Information Science, The Ohio State University, February 1974.
12. Stahl, F. A., "A Homophonic Cipher for Computational Security, *Proceedings of the AFIPS National Computer Conference and Exposition*, Vol. 42, AFIPS Press, Montvale, N.J., 1973, pp. 565-8.
13. Stahl, F. A., *On Computational Security*, Report R-637, Coordinated Science Laboratory, University of Illinois, Urbana, Illinois, January 1974.

RESEARCH IN THE PROTECTION OF PRIVACY OF PERSONAL INFORMATION DATABANKS—THEORETICAL AND TECHNICAL ASPECTS*—REIN TURN

INTRODUCTION

This two-year study, funded by the National Science Foundation Grant GI-29943, was completed in March 1974. The research was focused on a model of protector-intruder interactions in databank systems which identified the economic variables necessary for rational decisions, both by the protector of a databank and would-be intruders, in implementing protection systems or attempting to overcome these, respectively.[1] Within this framework the research was conducted in three areas: theoretical studies, system studies, and compilation of an annotated bibliography. The results of each are summarized below. Details are available in published papers and reports.

THEORETICAL STUDIES

We examined the following theoretical questions that arise in the context of protecting privacy and security of personal information databank systems.

Existence of uncrackable databanks

The question was, are there fundamental mathematical and logical principles by which it is possible to establish the existence of uncrackable (in the sense of Coates[2]) databanks—those which by their very nature would provide perfect privacy? We concluded[3] that if both the databank protector and the intruders have unlimited resources available, then no uncrackable (in the sense of Coates) databank exists; but if both the protector and intruder have limited resources, then uncrackable databanks may exist.

Applications of information theory

We established an information-theoretic model for data storage in computer files and showed how Shannon's com-

* The research described herein was supported by the National Science Foundation Grant GI-29943. However, any views or conclusions contained in this paper should not be interpreted as representing the official position or policy of the National Science Foundation or The Rand Corporation.

munication theory of secrecy systems and the rate distortion theory can be used to design irreversible privacy transformations for application in statistical databank systems.[4,5]

Data aggregation transformations

We explored the applicability of elements of Kolmogorov's ϵ-entropy theory for establishing measures of the degree of protection afforded by data aggregation in statistical databanks. Although a natural metric for databank purposes was defined, the calculation of ϵ-entropy or the ϵ-capacity of the ensemble of records has thus far been intractable (the problem is of the "sphere-packing" type) and without such a calculation this approach does not lead to useful results such as those of classical Shannon information theory.

Number-theoretic aspects of passwords

We examined the use of pseudorandom numbers as passwords in databank systems that have large numbers of remote terminals. We showed that various logistically attractive password generation and distribution systems are vulnerable to simple number-theoretic analyses that permit deducing the passwords of all the terminals. We proposed new password generation and distribution strategies to reduce such vulnerabilities.[6]

Modeling of protector-intruder interactions

We borrowed notions from game theory to set up an economic model of the "competition" between a databank protector and an economic profit seeking intruder.[1] This model suggests optimal policies for both. Before the model can be effectively used in practice, however, it is necessary to derive ways to measure the effectiveness of protection mechanisms, the value of information protected, the costs of both protection and intrusion, and the information loss. We have formulated strategies for deriving such measures.[7,8]

Centralized vs. decentralized databank systems

Many people intuitively feel that a centralized databank is a greater threat to privacy than a set of decentralized databanks. We have compiled sets of arguments, pro and con, on this question. Our score sheet shows that, if properly designed, implemented and controlled, a centralized databank should be able to provide more privacy protection than a decentralized databank.[9] This appears to be so for most of the threat scenarios that we have been able to invent.

A philosophical aspect of the privacy question

It is argued here that fear of the unknown and lack of knowledge by a person whose record is in the data bank of "What information on him is in the databank?" "Who is requesting information on him?" "What information is released to the requestor?" and even the fact that his record is in a particular databank, contributes greatly to his concern about data privacy. It is suggested that, although the Fair Credit Reporting Act is a step in the right direction, far more should be done to institute a sense of "reciprocity" between a databank's subject, and the databank's controller, custodian and users.

SYSTEM STUDIES

Another major part of the research dealt with system-oriented aspects of privacy protection and data security. Here we attempted to further the understanding of the nature of databank systems and their privacy and security problems.

Structure of databank systems

We established a model and a classification scheme for personal information databank systems which, we feel, effectively display the privacy and security aspects of such databank systems.[1,7] Along with this we established a catalogue of threats faced by personal information databanks and their subjects.[7,9]

"Model" security systems

We used the above classification to derive a series of representative data security systems for the various classes of databank systems.[7] We hope that such security system models can be used in practice as a framework for designing effective data security systems for specific databanks.

Privacy transformations

We analyzed the system implications of using (reversible) privacy transformations in protecting stored data and pointed out the pitfalls that face any uniformed attempts of their use—the level of protection achieved is often only illusory.[10]

Foundations of data security engineering

We outlined a general framework for the development of Data Security Engineering—a methodology and a set of techniques for designing effective data security systems in computerized databanks and information systems.[8] Such a methodology should include techniques for security requirements analysis as well as security system synthesis, derivation of measures of effectiveness and techniques for their practical evaluation, and tradeoff relationships between the various variables. A design methodology for data security systems is urgently needed. We hope to make the first necessary steps toward its development.

ANNOTATED BIBLIOGRAPHY

A major by-product of the project was the compilation of an annotated and cross-indexed bibliography on privacy and security related publications from 1970 through 1973.[11] The bibliography contains over a thousand entries. We also drafted a glossary of terms in the privacy and security area and a historical overview of the developments in the data privacy and security field. To date these have not been published.

FUTURE RESEARCH

We are planning to continue research in the data security area, concentrating on theoretical aspects of protection in resource-sharing computer systems, instrumentation of computer systems for real-time threat monitoring, and further development of the data security engineering methodology.

PROJECT MEMBERS

Principal investigators of this project were Mario L. Juncosa and Rein Turn. Other members of the project were Kathleen Hunt, Selmer M. Johnson, Irving S. Reed, and Norman Z. Shapiro. Martin Davis participated briefly as a consultant.

REFERENCES

1. Turn, R., and N. Z. Shapiro, "Privacy and Security in Databank Systems: Measures of Effectiveness, Costs, and Protector-Intruder Interactions," *AFIPS Conference Proceedings*, Vol. 41, Part 1, 1972 Fall Joint Computer Conference, AFIPS Press, Montvale, N.J., 1972, pp. 435-444.
2. Coates, J. F., *Computers and Privacy: Is There a Missing Theorem?* Note N-680 (R), Institute for Defense Analysis, Arlington, Va., April 1970.
3. Shapiro, N. Z., and M. Davis, *Uncrackable Databanks*, R-1382-NSF, The Rand Corporation, December 1973.
4. Reed, I. S., *The Application of Information Theory to Privacy in Databanks*, R-1282, The Rand Corporation, May 1973.
5. Reed, I. S., "Information Theory and Privacy in Databanks," *AFIPS Conference Proceedings*, Vol. 42, 1973 National Computer Conference, AFIPS Press, Montvale, N.J., 1973, pp. 581-587.
6. Johnson, S. M., *Certain Number-Theoretic Aspects of Access Control Passwords*, R-1433-NSF, The Rand Corporation, March 1974.
7. Turn, R., *Privacy and Security in Personal Information Databank Systems*, R-1044-NSF, The Rand Corporation, March 1974.
8. Turn, R., *Toward Data Security Engineering*, P-5142, January 1974.
9. Shapiro, N. Z., M. L. Juncosa, and R. Turn, *Privacy and Security in Centralized vs. Decentralized Databank Systems*, R-1456-NSF, The Rand Corporation, March 1974.
10. Turn, R., "Privacy Transformations for Databank Systems," AFIPS Conference Proceedings, Vol. 42, 1973 National Computer Conference, AFIPS Press, Montvale, N.J., 1973, pp. 589-601.
11. Hunt, K., and R. Turn, *Privacy and Security in Databank Systems: An Annotated Bibliography, 1969-1973*, R-1261-NSF, The Rand Corporation, March 1974.

HYDRA—A KERNEL PROTECTION SYSTEM*—WILLIAM A. WULF

The Hydra system is the 'kernel' base for a collection of operating systems designed to exploit and explore the potential inherent in a multiprocessor computer system. Since the field of parallel processing in general, and multiprocessing in particular, is not current art, the design of Hydra has a dual goal imposed upon it: (1) to provide, as any operating system must, an environment for effective utilization of the hardware resources, and (2) to facilitate the construction of such environments. In the latter case the goal is to provide a meta-environment which can serve as the host for exploration of the space of user-visable operating environments.

The particular hardware on which Hydra has been implemented is C.mmp,[1] a multiprocessor constructed at Carnegie-Mellon University. C.mmp permits the connection of (up to) 16 processors to 32 million bytes of shared primary memory through a cross-bar switch. The processors are any of the various models of PDP-11 minicomputers. Each processor is actually an independent computer system with a small amount of private memory, secondary memories, I/O devices, etc. Processors may interrupt each other at any of four priority levels; a central clock serves both for unique-name generation (see below) and broadcasts a central time base to all processors. Relocation hardware on each processor provides mapping of virtual addresses on that processor to physical addresses in shared primary memory.

The decision to use a 'kernel' approach, that is to provide only a set of mechanisms from which operating system facilities may be built, arose from two considerations: first, the conviction that only by this approach would it be possible to build a 'correct' system, and second, a desire to avoid predisposing the users of C.mmp to any particular mode of use through the idiosyncratic nature of 'the system.' (We want to learn how to effectively utilize a multiprocessor, *not* a multiprocessor plus particular operating system.) Thus our goal is to create an environment in which users can and will create their own operating environments—including scheduling and paging policies, file systems, etc.—and to support the simultaneous execution of an arbitrary number of such user-defined environments.

Given the general decision to adopt the 'kernel system' approach, the question remains as to what belongs in a kernel, and, perhaps more importantly, what does not. If a kernel is to provide facilities for building an operating system, and we wish to know what these facilities should be, then it is relevant to ask what an operating system *is* or *does*. Two views are commonly held: (1) an operating system defines a 'virtual machine' by providing facilities, or resources, which are more convenient than those provided by the 'bare' hardware, and (2) an operating system allocates (hardware) resources in such a way as to most effectively utilize them.

* This research is supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research.

Of course these views are, respectively, the bird's-eye and worm's-eye views of what is a single entity with multiple goals. Nevertheless, the important observation for our purposes is the emphasis placed, in both views, on the central role of *resources*—both physical and virtual.

The mechanisms provided by the Hydra kernel are all intended to support the abstracted notion of a resource (incarnations of a resource are called *objects*). These mechanisms provide for the creation and representation of new *types* of resources, as well as operations defined on them, protected access to instances of one or more resources within controlled execution domains, and controlled passing of both control and resources between execution domains. The key aspects of these facilities are the generalized notion of resource, the definition of an execution domain, and the protection mechanism which allows or prevents access to resources within a domain.

The mechanism used by HYDRA to achieve the various goals outlined above is an extension of the notion of a *capability* based protection system.[2,3] As with other capability systems:

(1) HYDRA supports the abstraction of a resource called an *object*. There may be several types of objects, typical examples of object types are pages, files, processes, and directories.
(2) For each type of object there are a finite number of operations, or *accesses*, defined on that object type. For example, operations such as 'open,' 'close,' 'read,' 'write,' and 'append' might be defined with respect to file-type objects.
(3) Objects are named by *capabilities*—which are protected by the system in the sense that a program may move them around, but may not create them or alter them in arbitrary ways. In addition to naming an object, a capability contains a specification of the accesses allowed to that object; that is, the capability determines the operations which may be performed on the object which it names. Hence, possession of a capability is prima facia evidence of the right to access an object in certain ways.

HYDRA extends the basic capability concept to allow any user to define new types of objects and operations on the new type. Thus the user may extend the basic resource types of "the system" as he sees it while not only retaining the protection applicable to previously defined resources, but extending the protection to the new resource. The user may wish to extend the system to provide a new facility (e.g., a different file organization), to alter management policies for a class of resources (e.g., a better disk request queueing algorithm), or to enforce a different security policy with respect to a resource. The extension of the capability model of protection involves several changes to the model:

(1) Objects may contain capabilities as well as data. (In the usual capability-based systems capabilities may only be kept in a special C-list associated with processes.) This permits existing objects to be used in the representation of new objects.
(2) The *operations*, or *accesses*, are defined by a special object type called a *procedure*. Since *procedures* are themselves objects, they may be manipulated, and protected, by the same mechanisms applicable to other objects.
(3) Protection is checked, as in other capability systems, when an operation is applied to an object; that is, when a *capability* is passed as a parameter to a *procedure*. In addition to this protection check, however, *rights amplification* may occur at this protection domain boundary (between caller and called). The called procedure may inherit *more* rights to the object than the caller possessed. This, in effect, allows the creator of a new object type to manipulate the representation of the objects of that type while preventing the owner of an instance of the type from doing so.

The HYDRA kernel is operational and is currently being used to explore the ease with which extensions of both function and security policy may be made in practice. To date our experience indicates that the HYDRA mechanisms can, indeed, be used to easily extend the environment of a user's program.

REFERENCES

1 Wulf, W. A., and C. G. Bell, "C.mmp—A Multi-Mini-Processor," *Proc. AFIPS FJCC 1972*, pp. 765-777.
2. Dennis, J. B., and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *CACM* 9, 3 (3/66), pp. 143-155.
3. Lampson, B. W., "Dynamic Protection Structures," *Proc. AFIPS Conf.*, Vol. 35, FJCC 1969.

A panel session—Accessing a data base through a minicomputer

SESSION CHAIRMAN—CHARLES A. LUPIEN

Combustion Engineering Inc.

Panel Members

R. D. Harrison, Jr.—Bell Laboratories
E. Lowenthal—MRI Systems Corporation
T. Richley—CIMCOM Systems, Inc.

OVERVIEW

Several speakers will address the notion of using small scale computing devices (or minicomputers) and associated data storage units as data base management "machines." These machines communicate with larger host processors to satisfy data handling requests emanating from programs running on the host. This is tantamount to removing the DBMS software from large main frames (e.g., IMS and total) and housing such software in one or more peripheral computers.

A panel session—Utility computing—A superior alternative?

SESSION CHAIRMAN—GEORGE J. FEENEY

General Electric Company

Panel Members

Robert D. Hilton—Coca-Cola Company
Robert L. Johnson—General Electric Company
Thomas J. O'Rourke—Tymshare, Inc.
Thomas E. Kurtz—Dartmouth College

OVERVIEW—GEORGE J. FEENEY

In 1905, approximately 50,000 small, privately-owned generating stations produced roughly half the total U.S. supply of electricity—largely for their own needs. Today, only about six percent of the nation's electric power is produced by such small local generating stations with the bulk of the country's power needs being met by some 200 large utilities.

The data processing industry is on the verge of a similar change for almost identical reasons. Despite the presence of some 60,000 widely-scattered and largely privately-operated general purpose computers, a trend away from in-house, do-it-yourself computing is becoming increasingly apparent. And, as in the electric utility industry, the driving forces are scale economy and variable cost aided by rapidly advancing technology.

Today, computer power generated at large, centralized facilities is transmitted and distributed over international data communications networks on a variable cost basis to most business centers in the free world.

This "Coming of Age" of utility computing brings with it a major turning-point in the industry, and broad implications for both users of computers and hardware suppliers. These implications and what they portend for business and data processing managers form the core of this panel discussion.

THE COMPUTER CONNECTION—A DIFFICULT QUESTION—ROBERT D. HILTON

Utility computing represents one of the most interesting, challenging and perhaps critical decisions facing many companies, large and small. The answer to the question is even more complex today with the confusion of integrated systems, data base management systems, telecommunications, minicomputers, etc.

Management, traditionally, has had great difficulty upgrading their needs and concepts to match an ever-racing evolution in computer technology. There have been no precedents for them to go by. There is a degree of "future shock" between management and systems technology.

In-house versus utility computing decisions frequently demand clear, long-range plans and heavy, sometimes irretrievable, commitments. Utility computing is not an easy question!

LARGE-SCALE USE OF UTILITY COMPUTING—ROBERT L. JOHNSON

General Electric Company's Switchgear Equipment Business Division, headquartered in Philadelphia, manufactures a wide range of equipment used by utility and industrial customers for electrical power distribution and control. Computer systems are used to support the work of every function and are vital to the success of the business. These applications include sophisticated approaches to production control, factory control and design automation, in addition to the conventional payroll and accounting work. Two years ago, the Division gave up operation of its only large-scale computer and elected to participate in a project with GE's Information Services Division to prove the feasibility of fully remote data processing (utility computing) on a mass scale.

This practical use of utility computing has proven successful, allowing the Division to significantly reduce data processing expenditures with no degradation in reliability or on-time job performance.

IMPACT OF UTILITY COMPUTING IN THE COMMERCIAL WORLD—THOMAS J. O'ROURKE

Essentially what we're telling the business manager is that he can buy the right to use a highly sophisticated computing/communication resource on a pay-as-you-use basis, with access by virtually any type of terminal device from anywhere in this country and overseas, accompanied by a rapidly growing pretested library of applications packages. The user also can run his own programs.

What's the impact? If we can show reliability, which we can, and the price is fair, there has to be interest, and there is. The rationale for our kind of service appeals to everyone who is concerned with costs but doesn't want to be concerned with additional hardware and support operations and management time and precious capital dollars on product development and sales.

The impact of our services on commercial business will grow and broaden because of the accelerating growth in suitable application packages geared to special needs. Also, because the economy and general business environment is tougher and tighter this year, we should be more attractive to more businesses than the option of committing hard dollars to upgrading or installing in-house systems. Additionally, the convenience and flexibility of being able to access a network such as our TYMNET makes possible modes of business heretofore impractical for many firms.

Just having the utility resource as an alternative may give rise to the most significant factor in commercial business—management now has an outside, objective benchmark against which to measure the cost and the efficiency of its present service resource. That may be the most revealing insight and benefit of all.

IMPACT OF UTILITY COMPUTING IN EDUCATION—THOMAS E. KURTZ

The main reason for utility (network) computing in education is that no single institution, however large and rich, can hope to provide its computer users with the enormous variety of special services they need. The issues here are more complicated than cost and efficiency; of special interest is the necessity for the utility (network) to provide small portions of services to many individual users, and thus to play the retail function. The kinds of special services include: data bases (in large numbers), special complex programs, exchange and transport of simpler programs, access to special language processors, and access to unusual hardware.

A panel session—Structured systems development

SESSION CHAIRMAN—JACK SHAW

Touche Ross & Co.

Panel Members

Kenneth W. Hunter—U.S. General Accounting Office
Gregory L. Brennan—The Chase Manhattan Bank

OVERVIEW—JACK SHAW

THE NEED FOR A METHODOLOGY

The management of data processing projects has yet to proceed from a loosely practiced "art" to a professionally disciplined science. EDP practitioners have not established a "body of knowledge" or a methodology or a standard approach to the development of systems. Such a body of knowledge is a prerequisite to the effective management of systems projects.

A systems management methodology will satisfy a number of "needs":

- A body of knowledge which may be taught to computer science and general management majors as well as EDP professionals.
- A basis for structuring the planning, development and implementation of systems projects.
- A vehicle for communication with general and executive-level management in order to achieve their commitment to and involvement in systems projects.
- A basis for planning and controlling systems projects.

METHODOLOGY CHARACTERISTICS

A methodology which meets the above needs should contain the following characteristics:

- The methodology should be independent of the nature of the project being developed.
- The methodology should have predetermining review and approval check-points and end-products which may be used as project documentation.
- The methodology should be non-technical and communicative to non-EDP management.
- The methodology should reflect the unique needs and operating environment of the organization involved.

APPROACH TO THE DEVELOPMENT OF A METHODOLOGY

The approach to the development and implementation of a methodology within an organization will and should vary with the nature and size of the organization and, most importantly, with the management style of the organization. The panelists will elaborate on the approaches used within their public and private sect or organizations.

BENEFITS

There are two principal thrusts to the benefits to be derived through the introduction of a standardized approach to systems development within an enterprise:

- To the individual—the professional will achieve a basis for communications with the user of his services and to the management which will appraise his performance. The professional will not have to rely on technical jargon or other unintelligible means to communicate progress or problems, but rather will become a part of the total organization. The professional will acquire a "body of knowledge" which may be built upon to improve his job performance and satisfaction.
- To the organization—the organization within which a standardized methodology is introduced will benefit not only in terms of the professional growth and satisfaction of its employees, but in terms of harnessing computer and systems technology to the needs of the organization. An appropriate "body of knowledge" for the planning, development, and implementation of systems projects will help assure that the organization is working on the "right things" and that planned projects are developed and implemented in a managed fashion.

KENNETH W. HUNTER

In the public sector, the leadership for developing standard methodology has been taken by the General Accounting Office. An initial study was conducted by the National Academy of Sciences for the General Accounting Office in 1972. The academy emphasized the need for principles and standards and proposed a broad range program for development. In line with the academy's recommendation, the General Accounting Office has created a Task Group on Principles and Standards for Computer-based Information Systems. The Task Group has members from Federal, state and local government and from the private sector. Their primary concentration in 1974 is on the cost control and accountability for the development and operation of computer-based information systems. The principles and standards, or good practices, developed by the Task Group are intended to provide guidance to system developers and system managers, and also to provide criteria for auditors and other independent evaluators of information systems developed and management. The GAO has closely coordinated its work with a GUIDE/SHARE project for the development of accounting procedures for the data processing function, including the methods of charging users.

A panel session—management impact of networks

SESSION CHAIRMAN—EINAR STEFFERUD
Einar Stefferud and Associates

MODERATOR—DAVID J. FARBER
The University of California at Irvine

Panel Members

Laurence H. Baker—The California State University and Colleges Office of the Chancellor
Joseph D. Naughton—National Institutes of Health, Bethesda, Md.
Ronald P. Uhlig—U.S. Army Materiel Command
John R. Lanahan—Inland Steel Company
Carl H. Reynolds—Hughes Aircraft Company
Leland H. Williams—Triangle Universities Computation Center

Technical feasibility of networking is putting pressure on management in education, government and industry to deal with some new but fundamental policy issues. There are a number of important management issues to consider.

First: Communications networks facilitate major sharing of computer resources across major organizational boundaries with the result that difficult new problems are being forced to the attention of management at all levels.
Second: Many organizations have grown dependent on their computing facilities. With networks this dependency is shifting toward foreign (or outside) computing facilities, causing power structure shifts which threaten management with rethinking their organizational structures.
Third: The management problems of sharing have not been solved in pace with technical developments. Sharing is now feasible but the management problems are unsolved. A substantial effort will be required to find some way for organizations to afford the risks of becoming dependent upon outside facilities, if networking is to become acceptable.
Fourth: The role of the technician must be fully understood in relationship to the politics of the power structure shifts that result from networking developments. We are all threatened by real dangers if technicians and managers fail to play their proper roles in the effort to solve problems posed by networks. It is vital that all the players in this drama understand their respective roles and avoid stepping on each others lines.
Fifth: The role of top management in evaluating or directing the use of networks must also be fully understood in relation to the need for organizational changes to accommodate the power structure shifts that result from networking developments.

A panel of experts from education, government and industry, who are currently facing and solving these problems will address the issues.

A panel session—The high cost of software—Causes and corrections

SESSION CHAIRMAN—RICHARD H. THAYER

Rome Air Development Center

Panel Members

John B. Slaughter—Naval Electronics Laboratory Center
Barry W. Boehm—Systems Group of TRW Inc.
Judith A. Clapp—The MITRE Corporation
John H. Manley—Air Force Systems Command
James H. Burrows—United States Air Force

OVERVIEW—RICHARD H. THAYER

In any large scale data processing development the cost of software development is three to five times the cost of hardware. It is estimated that in 1980's this development cost will jump to ten times that of the expenditures predicted for the hardware of that era. If the Government or industry is to maintain a ceiling on this cost, research must be conducted on ways to reduce costs in the various phases of software development such as design, specification, implementation, etc. For purposes of this session, "cost" will encompass not only the resources necessary to develop the software or transfer it, but also costs associated with delays in both delivery and "fixing," and resources wasted with software of poor quality. This session is an outgrowth of a tri-service/industry symposium entitled "The High Cost of Software" which was held at the Navy Post-Graduate Center, Monterey, California from September 17-19, 1973, in which service, industry and university experts attempted, for the first time, to get a better understanding of the costs of computer software and to make some recommendations as to what could be done about it. This session deals with the causes of software costs, the identification of both specific trouble areas and proposed R&D approaches that could be taken to eliminate the negative impact these areas have, and the particular ongoing research projects aimed at reducing software costs.

UNDERSTANDING THE SOFTWARE PROBLEM—JOHN B. SLAUGHTER

The so-called "software problem" is not only a very difficult one to solve but is equally difficult to define and describe. The problem has many facets; cost, quality, and timeliness of delivery, being the more obvious ones. Clearly there are strong interrelationships that exist among these factors, and any perturbation of one causes interactions with the others, which can cause uncontrollable reactions if careful management is not applied. Much can be learned by studying the development of hardware and attempting to transfer the tenets of successful hardware design to the software design process. Some progress is being made in this direction, e.g., "structured programming," but a generation of non-structured approaches to software development is difficult to eradicate completely. Certain key software problems can be identified, however, and there are some practical approaches to their solution.

SOFTWARE RELIABILITY AND SOFTWARE ERRORS—BARRY W. BOEHM

Roughly half of our efforts to develop software are spent in activities which attempt to guarantee its reliability. In spite of all this effort, however, unreliable software continues to cause numerous painful and embarassing operational problems. There is, however, a relationship between software reliability and software errors. Error analysis is most helpful in providing clues for how to *build-in* reliability (error prevention) and how to *improve* reliability (error detection and correction). However, without an underlying software reliability model, error analysis can tell us relatively little about *estimating* reliability. There is some body of quantitative knowledge of software errors: their general frequency, and their correction with various characteristics of the software development process. There is one study available concerning a detailed compilation of error type and an analysis of how these types of errors could be prevented or eliminated. Software reliability tools can be developed in the future that can prevent or eliminate the various types of software errors.

AUTOMATED MONITORING OF SOFTWARE QUALITY—JUDITH A. CLAPP

High software costs do not constitute a problem if those costs are reasonable for what is delivered, i.e., compared to the value of the services rendered by the software during the period of its use. Two aspects of software development have been most responsible for the dissatisfaction with software costs; delays in software delivery and poor software quality. Schedule slippages result in unplanned additional expenditures and loss of services. If software does not meet its requirements, fails under operational use, or cannot be maintained and modified easily then the price of the system is never justified. Better management tools and techniques are a key to maintaining schedules and improving software quality. Automated aids are needed to monitor progress and measure quality of a software product during its development.

EMBEDDED COMPUTERS—SOFTWARE COST CONSIDERATIONS—JOHN H. MANLEY

Software costs are uniquely impacted during the development of large scale systems containing embedded computer subsystems, such as automated rapid transit systems or modern aircraft employing digital avionics. When engineering such systems, management emphasizes the overall system often to the detriment of the integrating software. Too often the software must be cleverly forced into a hardware subsystem that has been suboptimized to fit mechanically into the overall system. Also, the task of integrating computer subprograms into a complete software package is enormously complicated by the requirement to also integrate numerous mechanical subsystems affected by the software. These and several other issues are identified as problems with embedded computer systems that increase software costs. Solutions to reduce the high cost of this specialized software are currently being developed by the Air Force for acquiring their large scale automated systems.

JAMES H. BURROWS

The "High Cost of Software" as a phrase is an attention getter. The "high" is provocative. I, for one, do not get turned on by such rhetoric. If what is meant by the phrase is that software related dollars are increasing both in size and in percentage of ADP budget, I guess I can only agree and say "what did you expect to happen," economy of scale of manufacturing large volumes apply only to part of the ADP budget. However, precise layout of the jobs to facilitate production line operation, a job done to facilitate such economies is comparable to a software job. The production process we are working toward is the production of data processing support for some operations and that does have large scale economy. In this we have both been responsive and economic, or we would not be supporting as many activities as we are today.

However, although I reject any idea that software *development* costs should decrease like hardware *production* costs, I do feel that the examination of the development process can lead us to areas for decrease in costs.

Due to the noticeable increase in ADP use and thus in process and computer program development costs, much well intentioned but sometimes misplaced effort has been spent getting the "process" under control. Most of the efforts copy good management practices of other development or production disciplines. The major effort has been "visibility" and "configuration control," both good but useless if administered by someone whose image is "production" rather than "development," not that there aren't some production aspects in development. The number of management overseers is legion and appears to be growing. This is a cost growth area we can avoid.

In point of fact, I feel that the foremost problem we have in reducing software costs is the shortage of talented discipline practitioners at both the bench and management level. Where "square fillers" are not enough, nor necessarily even useful.

Another difficulty is the changing and/or incompletely stated requirements. To a degree this is unavoidable, but can be expected and provided for. However, the number of midwives who are neither responsible for development, for operation and maintenance, or for use, but who have a jurisdictional interest and represent peripheral consideration, is again legion. Another cost to avoid.

There is much we can do in shortening and strengthening communication lines, orienting users in the new technology, choosing good people and facilitating their efforts with good tools, putting more formalism into our management and our development process so that we, *not others*, can see where we are and are going. What we can't stand are practitioners who are not fully competent, and management direction from people whose technical competence is in another field or another phase of effort. There is a world of difference between the overall concept of *development* and that of *production*.

A panel session—Effective use of computers

MODERATOR—MARVIN M. WOFSEY
George Washington University

Panel Members

James V. Milano—Pfizer and Company, Inc.
Phillip J. London
Robert L. Johnson—Babson College
Phillip C. Howard—Applied Computer Research

OVERVIEW—MARVIN M. WOFSEY

This panel focuses on the critical areas of evaluation in order to promote effective computer installations. Comparatively short papers will be presented on: evaluating computer installations, evaluating programs before they are put on the computer, reevaluating programs after they are running on the computer, and what management should know about the performance of their system. These will be followed by a panel discussion.

AUTOMATED DATA SYSTEM (ADS) DEVELOPMENT PLANS—PHILLIP J. LONDON

This presentation will discuss the structure of ADS Development Plans used in the management evaluation and monitoring of large scale ADS projects. The ADS Development Plan is intended to be a comprehensive, detailed justification and economic analysis of ADS development, conversion or major revision proposals. Preparation of such plans can significantly enhance management's capability of achieving ADS project goals.

The high cost of developing and operating large automated data systems, and their importance to the effectiveness of the organizations they serve, necessitates the use of a systematic procedure and standard discipline for justifying ADS from inception through full operation. The stated objective of such a framework is to assure that the ADS will: (1) perform as specified, (2) become operational at the time planned, and (3) not exceed the allocated cost.

A STUDY OF THE REEVALUATION OF APPLICATION PROGRAMS ONCE PUT ON THE COMPUTER—ROBERT L. JOHNSON

"For every benefit you receive," Emerson wrote, "a tax is levied." The development of computer applications gave business many benefits. At last the tax collector is catching up with us.

Our delusion that technology has the answer to everything has involved us in many unsuccessful EDP applications. These applications are now unsuccessful because the organization has failed to determine if the output is still practical, useful, and accurate. Installations have failed to realize that successful computer applications involve continual reevaluation as the organization develops. To assume management responsibility for a successful EDP application, management must think of its own posture in trying to establish correct control procedures.

Information is a non-consumable commodity. It grows endlessly, but the system builders spend little time in deciding when and what data or programs should be destroyed.

WHAT COMPUTER CENTER MANAGEMENT SHOULD KNOW ABOUT THEIR SYSTEMS' PERFORMANCE—PHILLIP C. HOWARD

The growing interest in performance evaluation and enhancement is a clear sign of management's growing awareness of generally poor system utilization and their interest in increasing the cost/effectiveness of the computer installation. A manager does not necessarily have to be expert in performance evaluation techniques and methodology, but he should insist on regular reports covering various aspects of system performance. Some basic and relatively simple indicators can serve to direct his attention to the areas that offer the greatest promise for improvement.

A panel session—Charge-out system for management acceptance and control of the computer resource

SESSION CHAIRMAN—RICHARD L. NOLAN

Harvard Business School

Panel Members

Charles Carey—Xerox Corporation
Michael J. Samek—Celanese Corporation
K. Sreenivasan—MITRE Corporation
John V. Soden—McKinsey and Company
Myron Uretsky—New York University

OVERVIEW

This panel focuses on the issues of using the pricing mechanism to both control and exploit the computer resource. The pricing mechanism for computer services and charge-out systems are employed primarily for control purposes. A more powerful role exists in the area of exploiting the computer resource and engendering user management acceptance and accountability. The panel will build upon ideas in the following paper in both the management and design issues of charge-out systems.

INTRODUCTION

The use of pricing, or familiarly charge-out, for computer services is one of the most unexploited management tools. On the one hand, this is rather startling since pricing has proved to be an effective tool for controlling resource allocation processes. For example, inter-divisional transfer pricing has been successfully employed to influence plant performance. Another example is the use of rates of return in capital investment. The use of computers in organizations can be likened to these resource allocation processes. On the other hand, the slight attention paid to pricing as a management tool is understandable in light of some of the tough practical problems involved in its use.

Recently, many of these practical problems have either been resolved or satisfactorily coped with enabling the pricing mechanism to serve the computer resource allocation and control process. Xerox is an example of one of the companies that is beginning to tap the potential of the pricing mechanism. Xerox has implemented a "standard cost" type system for processing computer-based applications. Each application was subjected to an industrial engineering study to establish its standard cost. The standard costs used market based prices for computer resources. The company then charges users the standard cost for processing their applications throughout the year and actual costs are also tracked and available to users on request. Where several users share the benefits of a computer application the processing costs are allocated between them. Users decide upon a "fair" appropriation of the costs. If one user decides to eliminate his report, the other users must pickup the added costs. The intent is to provide accountability to those that benefit from computer resources, and to provide feedback information so those that benefit can more effectively behave as responsible users.

The important point is that standard cost type systems for computer services are feasible and thereby offer an extremely powerful management tool for both controlling and exploiting the computer. Yet, the tough problems of designing appropriate systems for pricing and charging of computer services remain. Our intent is to identify these problems and issues in a manner that will permit examination of strategies and tactics for resolving them.

OBJECTIVES OF A PRICING SYSTEM

As a service function, the somewhat conflicting objectives of the Information Systems Department are to:

1. Assist in the full exploitation of computer resources throughout the organization—i.e., increase service
2. Provide computer services at the most economical level for the organization—i.e., control costs of resources.

Since the resource pricing mechanism should assist the Department in achieving its overall objectives, these two objectives are thus fundamental to the design of a specific pricing system.

MANAGEMENT DESIGN ISSUES

Once objectives of a pricing system are specified, we believe that there are five major management design issues that must be resolved.

1. *What computer services should be charged-out?*—The most obvious service to be charged-out is computer processing. However, the pricing strategy is confounded by the mixed bag of functions performed in many mature Information Systems Departments which include programming, systems analysis, management science modeling supervision, facilities, teleprocessing network usage, data entry, standards, forms and procedures, and messenger services.

One of the tougher issues is how to handle R&D type of computer services. Every company has potential applications of computers for which it is important to keep abreast of the emerging technology. Point-of-sale technology is a good example for retail stores. The issue is how to support the R&D effort cooperatively when no one user would be willing to bear the costs.

Computer data bases provided another dimension to this issue. Computer data bases offer a potential for shifting the system design philosophy of computer-based systems in a way to better use data. The technology is emerging and the question is what role should the charge-out system play and when.

2. *On what basis should computer services be charged-out?* A myraid of philosophies and interrelated bases exist for charging:

- market price versus actual costs
- cost recovery versus profit/loss
- average cost versus incremental costs
- standard costs versus actual costs
- controllable versus full costs

Although it is difficult to capture all the dimensions of these alternatives, the essence of them seem to be manifest in the type of responsibility center for the Information Systems Department, and their effect on user behavior. The responsibility center issue is usually posed with the following question: Should the Information Systems Department have profit-center responsibility or cost-center responsibility? Another way of stating the issue is whether the Information Systems department should be allowed to set market competitive prices for computer services or should they charge-out their actual costs. This issue often becomes influenced by the politics of the organization. For some managers, the idea of making a profit through providing an internal service seems inappropriate.

3. *What are the relative importance of key design parameters for a charge-out system?* There is almost an infinite number of ways that charge-out systems can be designed. The general problem is to trade-off the extent and sophistication of "effectiveness" design parameters such as fairness, acceptability, simplicity, accountability, timeliness, stability, comparability, and flexibility, with their implementation and maintenance costs.

If the user (or consumer) of computer services does not clearly see the effects of charging on his budget or organizational unit in general, then he won't be influenced by the charges. However, even if he clearly sees the effects, there must be the proper incentives for him to take action in an intended way. These attributes can be stated in the form of two questions for a charge-out system.

1. Does the charge-out system *communicate*?
2. Does the charge-out system *motivate*?

On the one hand, it is desirable to communicate the economic cost of the computer services of a user so that intelligent decisions can be made on the value of computer services relative to other resource alternatives. The economic costs of computer services include both the cost that the user directly control such as computer processing, and costs that he does not directly control such as operating system maintenance. Ideally, the user should effectively make decisions like whether a computer-based marketing information system would better serve his needs than adding more salesmen or increasing inventory stock.

On the other hand, it is desirable to provide the user with feed-back that tends to motivate him to reassess the value of computer-based systems, and which contributes to developing an overall awareness about the nature of computer-based resources. To accomplish this, it is important that the user is cognizant of an "action-response" phenomenon. For example, if a user cuts out one report, he should see some effect on his charge. If he cuts out the entire system, he should see, perhaps, a more significant reduction in his computer charges. During budget reviews, he should view computer-based system redesign as one alternative to becoming more efficient or effective. By holding the user accountable only for those computer-related costs that he directly influences provides an incentive for him to manage those costs in a similar manner to controllable personnel and facilities costs.

Charging for shared computer-based applications present a particularly difficult problem. Shared computer-based applications have similar properties to production processes which give rise to joint and by-product costs. For example, a perpetual inventory computer-based system has potential for producing market-oriented product activity reports for a low marginal cost for system design and programming. The question that arises is how much should the respective manufacturing department and marketing department be charged for processing the inventory control application.

Especially where the computer is involved, situations arise that are good for the company as a whole but which discriminant against one user in terms of cost incidence. For example, with emerging data base technology, it is generally unfair to levy the start-up costs of implementing data base technology to the first user. In order to prevent unfair discrimination against one user for decisions made for the good of the company, the issue of incentive pricing arises. Incentive pricing focuses on pricing computer services to effect a desired behavior on the part of the user. A familiar example of incentive pricing is to place a high price on every magnetic tape that is used relative to disk utilization in order to provide the user with an incentive to alter his programs to use disk files rather than tape files.

Many companies have struggled with and coped with the design issues of charge-out systems only to lose the confidence of users because of inconsistent charges for computer processing. The main problem has resided with multiprogramming environments, and now virtual memory environments. Stated simply, the problem is to charge the user the same cost when he processes the same job. Experience of charging the user significantly different costs for running the same job tends to destroy the user's confidence in the computer service department. Eventually, the loss of confidence leads to a situation where the user simply ignores his computer charges.

Attempts to solve the problem from a technical approach have also frequently ended up in failure. These attempts have engaged highly sophisticated and complex charging algorithms. The net result has been a charging procedure so esoteric as to defy virtually all comprehension by the user.

Changing computer configurations have resulted in additional complications in maintaining consistent charging. Computer system configuration changes move with advances in the technology and changes *should* be frequently expected. Unfortunately, configuration changes have created havoc with many charge-out systems by causing the charges to users to significantly vary. More often than not, the result has been a loss of confidence and a tendency for the user to ignore computer charges.

4. *How should the charge-out system be administered?* Central administration and policy-making are keys to effective use of the pricing mechanism for controlling and exploiting computer services. Yet there is a basic conflict for those Information System Departments that are given profit-center responsibility and must deal with user departments that are profit-centers. For the profit-center responsibility approach to work in concept, the profit-center managers must be allowed large latitude in their (discretion). One of these elements of latitude of discretion is to procure services and resources as efficiently as possible. If more efficient services are available outside, the reasoning goes that the profit-center manager should be allowed to procure the services outside.

The problem in practice is that it isn't easy nor straightforward for a user to pick up his computer-based application and take it to an outside service bureau. One problem arises from the fact that most applications are designed to run on unique computer systems and often use shared files. A second problem has to do with applications involving company sensitive information, and information private to the company's personnel.

Even if it is feasible for the user to take his applications outside, however, a basic issue is that the expenditure to the outside vendor is with "real" money. If the company has already committed to an investment in a computer facility and has the capacity to serve the user, the computer facility investment can be undermined. This not to say that special cases cannot arise where it is for the good of the company to use outside computer vendors. A familiar example of such a special case is the processing of large linear programming models. The issue turns on (1) how informed the user is on the implications of the "going outside decision," and (2) sound policies to maintain the interests of the company when those interests are in conflict with the interests of an individual user.

The steering committee has often been used to provide a forum for maintaining company interests and exploring policy-making alternatives. However, most companies have experienced disappointment with their EDP steering committee. Issues such as the role of the role of the steering committee, make-up, and authority remain outstanding.

5. *How should the charge-out system be used to ensure computer system efficiency?* In a somewhat oversimplified sense, the user if charged for the full cost of computer services, can go outside for those services, and does so, then there is prima facie evidence that the inside services are inefficient. Unfortunately, there are so many "ifs" involved that it is difficult to evaluate computer system efficiency in this manner.

There is another way. With minor adjustments to the accounting software integral to most operating systems, a great deal of information can be obtained about computer system efficiency. The information can be used for altering the computer configuration and system design techniques. A project carried out by the MITRE Corporation indicates that the IBM Systems Management Facility (SMF) for accounting of computer costs can be used to obtain much of the same resource utilization information currently obtained from hardware and software monitors.

PRAGMATIC FACTORS INFLUENCING CHARGE-OUT SYSTEM STRATEGY

The choice of pricing strategy is likely to be bound up in the transfer pricing or cost allocation policies of the host organization for other services. For example, if there is a corporate accounting and purchasing activity which does charge out its services to divisions, the corporate computer activity will tend to do likewise—whether rightly or wrongly.

The choice of strategy is also complicated by the organizational relationship of the Information Systems Department to the host organization. In other words, when the Information Systems Department is centralized, hybrid, or decentralized relative to its host, it creates organizational boundaries across which the strategy has to operate. In addition, there is a growing trend to establish computer services subsidiaries. This raises additional issues of equitable pricing for internal versus external users of these services.

The pricing strategy in all likelihood is also related to the basic economics of the industry of the host organization. For example, MIS executives in the banking industry can generally make persuasive arguments for dual processor configurations due to the high "cost" of not closing business during overnight processing. Thus, daytime processing in the banking industry has a tendency to be "freer" than in some other industries. Hence, in all likelihood, their pricing strategy will not tend to focus on dampening the interactive load during the daytime hours.

Finally, the choice of pricing strategy is related to the stage of development of the EDP effort in that the objective of the strategy will vary from one stage to the next—i.e., at one point, the objective may be to cultivate users and at another to control cost growth of user demand.

SUMMARY

Our position is that the use of computer services in organizations is a resource allocation process. One way to control the resource allocation process is through a pricing mechanism. We believe that the pricing mechanism, or charge-out approach, is a powerful management tool for both controlling the use of the computer resource and engendering management acceptance of it as an exploitable resource. We also are aware of the difficulties in tapping the charge-out mechanism. Our purpose is to identify the main design issues that must be resolved and coped with in order to incorporate the pricing mechanism into the process of mining the computer resource.

A panel session—Accelerating information delivery

SESSION CHAIRMAN—DON S. CULBERTSON

Argonne National Laboratory

Panel Members

Dennis Elchesen—Lawrence Livermore Laboratory
Miss Mary Ann Swanson—Evanston Township High School
Frederick G. Kilgour—Ohio College Library Center
Dr. Russell Shank—Smithsonian Institution

Purpose: To describe using three example institutions, the potential size and scope of the library market for computers, peripheral units and specially designed software. There are literally thousands of libraries in the U.S.; public school, special, college, medical and law. Until recently only the largest institutions with large computer facilities and foundation grants have been able to develop the custom made systems to tend to the library housekeeping, handle enormous inventories of books, index files of journal articles, notify users of items available as fast as they are published, and search the small segment of the world's knowledge which is in machine readable form.

The Special Libraries Association program illustrates three aspects of the development of library automation from the beginnings noted above.

Part 1. A large specialized library developed to a high degree of sophistication.
Part 2. A small library able to take advantage of the computer hardware and systems staff of its parent organization (which itself may not be very large). Interest and opportunity lead to a library highly automated for its type.
Part 3. A consortuim of various sized libraries receiving online services from their center. Most of the smaller members could not have afforded the system development costs.
Part 4. Question moderator and program summary.

A panel session—Certification of computing personnel: prospects and potential impact

SESSION CHAIRMAN—FRED H. HARRIS

University of Chicago Computation Center
Chicago, Illinois

Panel Members

Robert N. Reinstedt—The Rand Corporation
Donn B. Parker—Stanford Research Institute
William J. Horne—Boston College
H. R. J. Grosch—Computerworld

CERTIFICATION OF COMPUTING PERSONNEL—PROSPECTS AND POTENTIAL IMPACT—FRED H. HARRIS

Sociologists have identified key attributes of professionalism by studying vocations which are widely recognized as professions. These include, among others, a high standard of skill and knowledge; public reliance upon the standards of its practitioners, and the observance of an ethical code.

Moreover, a look at the history of the traditional professions, such as medicine and law, as well as younger groups such as accounting, has yielded identifiable trends which mark a vocation as an emerging profession. For example, ongoing activities for the development of organized, formal training at advanced levels and associated testing requirements, usually fostered from within organized groups of practitioners, denote a change from trade to profession. Another such activity is the establishment, and peer group enforcement, of codes of conduct.

Thus, major segments of the computing industry's work force can be considered to be an emerging profession. But trend lines are not conclusive nor well established. Key questions remained to be answered, and the panel today will focus its attention on certification. Do we wish to be? Do we have the tools to do so in an effective way? What will be the impact as we proceed?

TECHNIQUES OF EFFECTIVE TEST DEVELOPMENT—ROBERT N. REINSTEDT

There are established acceptable guidelines that must be met in developing tests for general use. If these guidelines are not adhered to, the tests will not be able to gain acceptance, and if challenged will be found to be non-usable.

Four areas are of prime concern to test developers today. Reliability, validity, non-compromised tests, and tests that are free from cultural/ethnic bias.

Although actual construction of tests and test questions may seem considerably more complex, and refining the instrument requires tedious and sophisticated attention, the four areas mentioned above are the real crux of test development.

Because of the importance of these criteria, professional psychometricians have incorporated methods aimed at satisfying the criteria as a standard function of test development.

As tests and testing meet with more opposition, and are being questioned as never before, it becomes essential to produce tests that do exactly what they purport to do and do it equitably for the entire population exposed to the testing procedure. This holds true whether one speaks about tests used for selecting college students, graduate school admittance, job selection, certification or any other form whereby lives of individuals are affected as a result of the outcome of some type of examination designed for general use.

PROFESSIONAL CERTIFICATION AND THE IMPACT ON THE INDIVIDUAL PRACTITIONER—DONN B. PARKER

Certification is one more step in providing the practitioner with the feeling that he really belongs to a profession. He is separated from the identity with technicians and with other professions. But we can ask what is the size and range of performance of this profession when it is finally achieved? Will most programmers be relegated to technician levels? Will application programmers identify with their applications areas as higher level languages reduce the need for today's programming expertise? The shift to special interest groups within ACM and the Computer Society may be an indication of this. What will a certification requirement for membership in ACM do to its total membership? It doubled the size of the British Computer Society overnight. Will certification in the future have little or no more impact than it currently has? Will licensing requirements be legislated and change this situation? Why be certified?—Lots of questions to be answered only by trying it.

CERTIFICATION AND ITS IMPACT ON MANAGEMENT—WILLIAM J. HORNE

We define management in the usual textbook manner as those activities which involve "planning, organizing, actuating, and controlling." I prefer the less formal definition: "the art and science of getting meaningful work done through people." Within this latter definition, I believe, we shall find the major impact on management that certification of computer professionals will come to have.

Currently, management really has no definitive 'scientific' method for determining the quality and capacity for meaningful output of computer-oriented people. The present method, then, is to try them—the practitioners—one after another, until at least a satisfactory performer is found.

For certification to successfully and beneficially impact management, it must identify uniquely the above-average programmer, the above-average management scientist, the above-average information systems manager, etc., especially in terms of productivity.

Management desperately needs computer specialists who recognize the central economic issue of scarcity. It desperately needs computer specialists who "feel," with the behavioral scientist, the importance and significance of the human element with all its ramifications.

If certification succeeds in providing a handle on choosing people with these characteristics, it will affect management practices profoundly in the next several decades. We shall address this impact as thoroughly as time allows.

IMPACT OF CERTIFICATION ON THE COMPUTING INDUSTRY—H. R. J. GROSCH

For many years we have seen youngsters recruited out of academic dens of computer science and put to work in systems programming teams, with no evidence of practical know-how and no demonstrated *general* ADP skills. It will be greatly beneficial to the software industry, and to the software part of the hardware industry, to have a possible filter.

At the sales end, vendors are always being asked to recommend personnel to customers. Certification makes the process safer. And in a venue as diverse as the computer industry, a common experience, a shared area of discourse helps internal organization and understanding.

A panel session—Large information processing networks: development and operational experience

SESSION CHAIRMAN—HOWARD FRANK

Network Analysis Corporation

Panel Members

Davis McCarn—National Library of Medicine
Richard Sprague—Payment Systems Inc.
M. U. Ayres—Boeing Computer Services
John Perra—Aetna Life & Casulty
George Feeney—General Electric Company
Morton Blanchard—National Association of Securities Dealers

OVERVIEW

By 1980, 70% of all computers may be connected to terminals via extensive communication facilities. Information processing networks will influence nearly every segment of our economy with applications ranging from remote computing to credit verification and electronic funds transfer. The builders of these new networks will be faced with a host of problems such as selecting new devices, vendors, and common carrier offerings and operating systems in environments where multiple component failures are common. This session, whose panelists are experts in the art of making networks work, will address the problems encountered in building and operating large networks.

A panel session—Digital communications on cable systems

SESSION CHAIRMAN—WILLIAM F. UTLAUT

U.S. Department of Commerce

Panel Members

Peter M. McManamon—U.S. Department of Commerce
Hubert J. Schlafly—TelePrompter Company
Friend Skinner—MITRE Corporation
S. S. Tyler—Singer Company

OVERVIEW

Three technologies—digital transmission, computers, and cable systems—offer opportunities to meet the growing and diverse needs for high volume and rate of data and information transfer required by society for business, education, entertainment and crime reduction. Panelists will discuss technical and economic factors, and their priorities, associated with integrating these technologies, and others, to provide services such as electronic mail, banking, retail marketing, work and learning at home, surveillance and others.

A panel session—Information systems for ambulatory care

SESSION CHAIRMAN—ANTHONY I. WASSERMAN

University of California

Panel Members

William Cass—Massachusetts General Hospital
Jerome H. Grossman—Massachusetts General Hospital
Alfred H. Garratt—HEW
William V. Glenn, Jr.—Massachusetts General Hospital
Gerald A. Giebink—Health Care Management Systems

OVERVIEW—ANTHONY I. WASSERMAN

Until recently, most medical information systems were developed for use in hospitals, despite the fact that the majority of health care is given in clinics and physician's offices. There are several reasons for this situation, including the following:

(1) the nature of acute care implies a more urgent need for communication of information among health professionals;
(2) there is a greater volume of data collected on inpatients than on outpatients;
(3) there are larger sums of money per encounter involved, tending to make information systems more economical.

Within the past few years, however, the cost reduction in computer hardware, the increasing volume of paperwork in the medical field, and the advent of very large clinics have combined to make development of information systems for outpatient settings much more feasible.

Several such systems have become operational and more are presently under development. The earlier systems emphasize medical information and contain a subset of the patient's medical record, including a summary of the patient's most recent visits, a list of current problems and diagnoses, a list of current medications, and the most recent results of any laboratory tests. As the systems have become more sophisticated, more information of an administrative nature is being added, including patient visit scheduling, economic analysis of ambulatory care, utilization and membership statistics for clinics, and quality of care assurance.

There are a number of problems which must be overcome in the design of information systems for ambulatory care, including:

(1) disagreement over the content and format of the abbreviated ambulatory medical record;
(2) creation of a system which is cost-effective in terms of reduced provider time, hardware costs, software development costs, system reliability, and ongoing maintenance costs;
(3) development of a system which is flexible enough to evolve gradually as information needs and computing technology change;
(4) design of a man-machine interface which is acceptable to the providers who will use the system;
(5) difficulty in satisfying both medical and management needs in an information system.

Successful implementation of these outpatient information systems can result in improved quality of health care through better continuity of care and a superior ability to practice preventive medicine, while at the same time providing a management tool which can effectively control the costs of medical information processing.

A panel session—Terminal hardware and methodology in hospital information systems

SESSION CHAIRMAN—STANLEY E. JACOBS

American Hospital Association

Panel Members

Lou Phillips—Chairman of the Board, Medelco, Inc.
Marion J. Ball—Director, Health Sciences Center Computer Systems, Temple University
John W. Anderson—Administrator, Tabernacle Community Hospital and Health Center

OVERVIEW—STANLEY E. JACOBS

Opening comments will define the term "hospital information systems" (HIS). Two levels of HIS will be described—first the level of HIS wherein the computer is used primarily as a data collection and message switching device; second the level where the computer also maintains a data bank of patient information and assists in the scheduling and controlling of activities of the hospital.

Today there are approximately twenty vendors of HIS. Each vendor utilizes terminals unique to HIS system. Qualitative differences exist not only in the hardware, but in the data entry strategies and in the types of personnel for whom the terminals are intended.

Assisted by visual aids, there will be a description and discussion of the terminals of each vendor and the methodologies utilized for data entry and retrieval. Finally, a hospital administrator will discuss his ideas concerning the value of HIS to the short-term, general hospital.

Dr. Jacobs will describe the needs for HIS, the HIS objectives, and the features which enable systems to achieve the HIS objectives. Vendor system versus in-house development will be discussed. The rapid entry of vendors into the market-place will be pointed out, together with similarities and differences between vendor systems. Finally, a distinction will be made between the "data collection/message switching type of HIS" and the "total HIS".

A panel session—Computer algorithms for analyzing patient data

SESSION CHAIRMAN—JONATHAN CLIVE

Duke University

Panel Members

Lee B. Lusted—University of Chicago Medical School
Casimir Kulikowski—Rutgers University
T. Allan Pryor—Latter-Day Saints Hospital
Discussant—Bruce McCormick—University of Illinois-Chicago Circle

OVERVIEW—JONATHAN CLIVE

This topic deals with the computer-assisted evaluation of the questions of which several alternative diagnostic states might apply to a particular patient. We are also concerned here with the choice of therapy once a disease presence has been established. The computer-assistance scenario of this type must take into account both the types of diseases for which it is being instituted and possibly large amounts of patient data.

AN OVERVIEW OF COMPUTER-ASSISTED CLINICAL METHODS—LEE B. LUSTED

The clinical process often requires the physician to make decisions based upon a large volume of diverse information. Whether these decisions are of a diagnostic, therapeutic, or prognostic nature, the utilization of computer-assisted decision-making techniques can be of substantial service to the clinician. This discussion considers the general requirements of a computerized probabilistic information processing (PIP) system within the clinical framework, and directs attention to the following points:

(1) Accessibility and utility of the PIP system in terms of physician-computer interaction;
(2) The use of subjective diagnostic probabilities as input for the PIP system;
(3) Use of a PIP system for patient-management and choice of therapy.

A SYSTEM FOR COMPUTER-BASED MEDICAL CONSULTATION—CASIMIR KULIKOWSKI

To feel confident in using a consultation program, the practitioner must be able to understand the methods underlying the computer's decision-making capabilities, and know the scope and structure of medical knowledge upon which these methods rely. In some fields of medicine a structure of statistical association among clinical findings may be appropriate. However, a consultation program that attempts to include the knowledge of expert consultants in a field must include information on the anatomy, physiology, pathology, and therapy of the disease groupings for which it is designed to be effective.

A major problem consists of structuring this knowledge so that it can be used efficiently for diagnosis, prognosis, and therapy. It is unusual for traditional biomathematical models to be applied directly as a basis for clinical decision-making because of their excessive detail appropriate to the laboratory but not to clinical situations. What is required is a descriptive model of disease processes that can serve as a basis for the logical interpretation of clinical findings, supported, when possible, by more detailed models of physiology and function. A consultation system based on a descriptive, associational, and causal model of disease has been developed and applied to problems of glaucoma. The system provides recommendations on the management of patients, together with explanations of its reasoning in terms of the model of disease. Alternative models can be developed to incorporate differing opinions and emphases of specialists in the field giving the system flexibility and greater scope of interpretation.

A panel session—Computer-simulations of aspects of the diagnostic process

SESSION CHAIRMAN—MAX A. WOODBURY

Duke University

Panel Members

Richard Friedman—University of Wisconsin
David Gustafson—University of Wisconsin
Jonathan Clive—Duke University
Discussant
Max A. Woodbury—Duke University

OVERVIEW

This session discusses research related to study of the diagnostic process using computer-assisted methods, and from a perspective that is not necessarily disease-specific or decision oriented. Some of this research concerns itself with the detection of latent diagnostic and prognostic groups, as well as the study of diagnoses evaluated by physicians on computer-simulated patients.

A panel session—The use of computers for instruction and administration in elementary and secondary education

SESSION CHAIRMAN—SYLVIA CHARP

School District of Philadelphia

OVERVIEW

The use of the computer in education, especially in elementary and secondary education, has already made an impact. The number of school systems using computers for both administrative and instructional applications is growing. Questions, however, are still being raised, which require sharing of experiences by educators. When and for what applications can the same hardware be used for both administrative and instructional purposes, when and is it economical to buy time from time-sharing company, or when and how best to use stand-alone minicomputers require careful study. The objectives and needs of the individual school user or school system must be stated and then specifications given to the manufacturers so that these objectives can be met as manufacturers become more and more aware of the many uses of computers, especially in instruction, hardware may be built which satisfy the particular needs of education. The problem of training for administrators and teachers cannot be overlooked. Administrators and teachers need to know how to use effectively all the additional information made available. Also, the changing role of the teacher as the computer becomes the dispenser of information, needs to be understood. Manufacturers, educators, regional laboratories, private companies must all work together in this endeavor.

A panel session—University computer curricula

SESSION CHAIRMAN—DICK B. SIMMONS

Texas A&M University

Panel Members

Edward J. McCluskey—Stanford University
Aaron Finerman—State University of New York
M. L. Dertouzos—Massachusetts Institute of Technology
Jurg Nievergelt—University of Illinois

OVERVIEW

The status of university computer curricula will be discussed and recommendations of how university computer curricula can be improved in the future will be presented. The panel will discuss how universities can better meet critical needs of industry, business, and government in the various areas of information processing. With the advent of minicomputers, on-line computing, computer aided instruction and the rapid growth of data processing, the emphasis in computer curriculum has changed and some of the earlier ideas should be updated.

A panel session—Computer education for managers

SESSION CHAIRMAN—JAMES E. OBERG

DOD Computer Institute

Panel Members

Jack Butler—DOD Computer Institute
Bernard F. King, Jr.—IBM Corporation
Stephen Ruth—U. S. Navy

OVERVIEW—JAMES E. OBERG

Data processing is still a specialized trade, carried out by specialists. They talk to each other in a private jargon. They do not as a rule encourage outsiders to try to understand their craft.

For computer use to change from a craft practiced by a select guild to an aspect ot twentieth century technology available to all (like telephones, television, automobiles), the concepts of computers must be made understandable to the non-technical people who need computing power.

Up to now, computer education has concentrated on university training curricula for future members of the guild. For the rest of the population, their education—or mis-education—comes from newspapers, science fiction movies, telephone billing errors, etc.

Today, all kinds of people are face to face with computers. Their efficient utilization of available computer power depends on how well they understand what computers can and cannot do.

Standing between them and this efficient utilization is a "computer mystique," a modern mythology of the machine. This mystique, with its misconceptions and mysteries, is the result of the popular press, movies, and real world experience with computerized systems. The "mystique" was also to a large extent encouraged by members of the computer fraternity themselves.

Computer power can be extremely useful in the difficult and crucial problems faced by decision-makers. Managers and executives can use computer information systems to help get the right information at the right time to enable them to make the right decisions. The manager has got to allocate his resources among contending demands to get the optimal results in the end.

Efficient computer use in management has suffered from many factors. For example:

1. Misconceptions about the capabilities and limitations of computer systems.
2. Not enough attention to the information system which is to be serviced by the computer.
3. Not enough attention to the organization and human implications of computer systems.
4. Naivete regarding the work and talent required at all organization levels to develop, design, and maintain a truly effective computer system.

Misconceptions can be based on an overestimation of a computers power, or a refusal to concede that a computer has any real application in the "real world." Hope, fear, love, hate, the entire spectrum of human emotions all have been directed toward computer systems.

How can the functional manager in executive and administrative positions learn to use the power which computer systems can give him? This session will examine several introductory orientation educational programs now in existence. Their purpose is to educate the non-technical functional manager with regard to computer potentials for his organization.

COMPUTER EDUCATION FOR ADMINISTRATIVE MANAGERS—JACK BUTLER

Background—The Defense Department established in 1964 the DOD Computer Institute (DODCI), whose purpose was to conduct computer orientation and introductory courses for government executives who had no previous computer experience. DODCI has a staff of 25 instructors and is located in downtown Washington, D.C., in the Washington Navy Yard. Courses are offered both in Washington and on-site where any government installation requests them. More than 3000 students per year receive their first serious computer orientation from DODCI.

Objectives—The two-week orientation course is designed to provide an educational background for high level management personnel, both military and civilian, who have responsibilities involving general-purpose digital computer systems but who have had little or no previous orientation in automatic data processing.

The specific course objectives are:

A. To provide knowledge of the fundamentals of computer hardware and software, including their characteristics, capabilities and limitations.

B. To introduce essential computer system development concepts with related planning considerations.
C. To familiarize the student with the necessary computer and ADP terminology in an effort to close the communications gap between management and the computer specialist.
D. To introduce basic quantitative techniques and their application in the area of management science.
E. To provide a basis from which the student can pursue further individual study.

Course Description—The course covers computer capabilities, limitations and applications to include key concepts and planning factors for instituting new computer systems or improving existing systems. Each course provides computer "hands on" experience through laboratory work with a small digital computer, and with computer terminals in a time-sharing environment. Guest lecturers complement the theory and principles taught by the instructor staff. Upon conclusion of the course, the student has a sound basis upon which to add further computer knowledge, and he can well appreciate the potential of today's digital computer systems.

Results—DODCI's experience with education of non-technical management personnel indicates that two weeks of full-time instruction and lab work is indeed sufficient to overcome the "computer mystique," the psychological barrier which is the main cause of inefficient use of computer resources today. Students are enthusiastic about the course. The demand for this service among all branches of the military and government continues to grow.

EDUCATING THE NON DATA PROCESSING EXECUTIVE—FACT AND FANCY—BERNARD F. KING, JR.

The proposed 45 minute lecture is based upon our past two years of experience in educating IBM customer executives. The dimension of the effort during that period was 2500 executives who attended 10,000 days of instruction.

During that period radical changes were introduced in executive education. These changes reflected the pressure that on-line, interactive, data base system appear to impose upon an organization. Specifically the major topics are: (a) organizational planning (b) strategic management role (c) user management/DP management contention (d) technology in the organization. The lecture will explore why these changes occurred and the instruction content.

Throughout the lecture, the older Executive Computer Concepts education will be contrasted with the newer approach.

Finally a control group of 15-20 chief executive officers will be followed from completion of their education program through the next 6 months to measure the actual benefit, if any, that occurred to the organization for a one week investment of the CEO's time.

A BUSINESSMAN'S APPROACH TO THE DETERMINATION OF THE VALUE OF EXECUTIVE LEVEL EDP COURSES—STEPHEN RUTH

This presentation defines some of the major evaluative issues which are involved in executive courses in EDP technology. Many of the traditional measurements for effectiveness of an adult educational experience are found to be unsuitable. The major emphasis is on clarifying what should be the goals of a course of this type and, more importantly, an elaboration of some proposed methods for evaluating whether goals are in fact achieved. One aspect of this discussion which is somewhat counter intuitive is the fact that an executive's satisfaction or pleasure with the course, its surroundings, and its instructor frequently constitute a neutral and occasionally an inverse indicator of the real value of the course. Data based on a factor analysis technique is presented to develop this point.

Since the use of these courses is an expensive cost element in executive development and education, a cost-based evaluation system, which takes into consideration the real, as opposed to apparent, value of executive EDP courses is proposed.

A panel session—C-COMP—COPICS—Communications oriented manufacturing plan: "A paper-less factory approach"

SESSION CHAIRMAM—THEODORE A. BAKALAR
IBM Corporation

Panel Members

Theodore A. Bakalar—IBM Corporation
Dennis Sears—IBM Corporation

CONTROL of manufacturing shipping schedules and costs implies there is control over the manufacturing process. Very often production control does not have the capability to control the execution of the manufacturing plan. Lack of control is largely because of the information environment that exists. Two problems are very apparent—the vast amount of paperwork on the plant floor and the untimeliness of its information. A plant floor communication system can help solve these problems.

The "shop packet" is YESTERDAY'S roadmap!

This session will take a close look at C-COMP (COPICS—Communication Oriented Manufacturing Plan) as a new systems approach in the areas of receiving, inspection, material handling, stockroom, order release, labor reporting and dispatching to address the paperwork problems and provide a tool to help control the manufacturing process.

A panel session—Problems, perils and promises of computer graphics

SESSION CHAIRMAN—LAWRENCE ROSLER

Bell Telephone Laboratories

Panel Members
James D. Foley—University of North Carolina
John B. Macdonald—Western Electric Company
H. G. Marsh—Raytheon Company

OVERVIEW

For many years, profitable large-scale use of computer graphics has been "just around the corner." The panelists, representing industrial users, system developers, and university researchers, will present divergent views on what is required for the promises finally to be fulfilled. These views range from coding in tailored machine language to writing transportable programs in a high-level language. An open discussion, with audience participation invited, will follow.

A panel session—Numeric control machine tool technology and applications

SESSION CHAIRMAN—DR. JAMES WARNER
Northern Illinois University

Panel Member
Dr. Ronald L. Boase—CAM-I, Inc.

PART FAMILY CODE?—DR. RONALD L. BOASE

The problem of constructing a part family code for identifying parts having "similar" process plans is considered. The basic emphasis is on the role the part family code plays in relating descriptive information on the part to its process plan. The use of the typical digital structure of a part family code restricts the ability for representing this relationship accurately. Self-learning computer algorithms which express this relationship in functional form replaces the need for a part family code. Code identifiers labeling parts having similar process plans can then be more beneficially used.

A panel session—Manufacturing information systems

SESSION CHAIRMAN—EARL GOMERSAL

Motorola, Inc.

OVERVIEW

New eras of increased productivity and profits through advanced Manufacturing Information Systems—the great nonsequitur. Far from fulfilling years of advanced publicity and still resistant to new technologies, more economical devices, and a host of new optimizing gadgets, total systems answers to manufacturing problems continues to be the elusive "brass ring" for the adventurous and a Waterloo for those who have been given the resources to make it happen.

Dealing with manufacturing imperfections requires imperfect solutions. If that fox is not lost in the hunt, the next decade can be a promise fulfilled. Manufacturing Information Systems need not be limited to "mechanized tradition." A down to earth analysis of "do differentlys" will be presented.

A panel session—Shop floor control

SESSION CHAIRMAN—J. O'TOOLE
Mid America Corporation

Panel Member

James A. O'Toole—Mid America Corporation

SHOP FLOOR CONTROL—JAMES A. O'TOOLE

Controlling the factory environment will significantly increase productivity. Efficient coordination of jobs with man power and machines increases output and income. The Computer, if it is properly used, provides the capability for coordinating the elements of the factory floor more efficiently than manual systems.

Mr. James A. O'Toole, Vice President of Mid-America Computer Corporation, will address this subject in a practical manner based on installation experiences.

A panel session—What manufacturers would like to see happening in point-of-sale

SESSION CHAIRMEN—GERALD T. MONTGOMERY

J. C. Penney Company

and

VERNON L. SCHATZ

Jewel Companies, Inc.

Panel Members

Michael McHale—UNITOTE
Samuel Harvey—Singer Business Machines
M. G. Tomlin—IBM
Richard Simon—Sweda International
Joseph Vale—American Regitel
Richard Fried—National Cash Register
W. E. Carey—IBM
John King—Electronic Store Information Systems
Ralph Canada—National Cash Register
John Ineson—Singer Business Machines
Terry White—Sperry-UNIVAC
Fred Bialek—National Semiconductor Corporation

OVERVIEW

Implementation of electronic Point-of-Sale (POS) Systems is still in its early stages, so many of the potential benefits are still untapped.

This is one of four sessions (two each in Retailing and Distribution) in which POS manufacturers will describe the approach they have developed and the improvements in store operations their systems will make possible.

These sessions should be of interest to anyone concerned with the future direction of POS.

A panel session—Point-of-sale systems for supermarkets

SESSION CHAIRMAN—VERNON L. SCHAATZ

Jewel Companies, Inc.

Panel Members

Vern Sdhaatz—Jewel Companies, Inc.
Doug Brookings—Schnuck Markets, Inc.
Dan Minter—SAMI
Ron Low—Jewel Companies, Inc.
Lee Paulson—Allied Supermarkets, Inc.

OVERVIEW

The session on Point-of-Sale Systems will be primarily concerned with supermarket operations. It will emphasize live experience in the use of such systems. Discussions of current experience will be complemented by discussions of the anticipated impact of the Uniform Grocery Products Code (UPC) and scanning.

A panel session—Transferability of government information systems, problems and solutions. Is it cost effective?

SESSION CHAIRMAN—VERNE H. TANNER, JR.

NASIS

Panel Members

James J. Trainor—Department of H.E.W.
Nelson A. Howell—NASIS
Charles D. Trigg—IBM Corporation
Harold O. Casali—Department of Finance and Administration

OVERVIEW

One of the most significant challenges facing state information coordinators today deals with the ability or lack of ability to make use of the proven concepts of systems developed in other jurisdictions. The panel will attempt to develop a definition of the word "transferability," to outline the restraints and pitfalls which prevent a successful transfer and to suggest ideas and methods which can avoid these problems. While specific examples presented by the panel will deal with state activities, their problems and solutions should be of importance not only to all levels of government on a horizontal plan, but also to all levels of government on a vertical plan.

A panel session—Federal activities in information processing

SESSION CHAIRMAN—JOHN GENTILE

Assistant Postmaster General—U.S. Postal Service

Panel Members

Kenneth W. Hunter—U.S. General Accounting Office, General Management Studies Division
Clark R. Renninger—U.S. Department of Commerce, National Bureau of Standards
Gordon Yamada—Office of Federal Management Policy

OVERVIEW

Discussion and presentation of responsibilities of various federal agencies as they relate to the federal establishment, intergovernmental relations and the private sector in the area of information processing, especially that of the General Accounting office, the General Services Administration and the National Bureau of Standards in carrying out their missions.

A panel session—Law enforcement—Do the systems really provide the information and safeguards promised?

SESSION CHAIRMAN—GLEN E. POMMERENING

U.S. Department of Justice

Panel Members

Howard Bjorklund—Department of Justice
Melvin F. Bockelman—Kansas City Police Department
Alan A. Hamilton—REJIS

OVERVIEW

Law Enforcement—An updating of the advances made in providing information for the management of law enforcement and problems still facing these activities and the resources being brought to bear on their solutions.

Over the past year several new and very successful information systems have come "alive" throughout the country. While at the same time questions have been raised as to whether the systems have really provided the type of information required and/or contained the safeguards which were promised.

A panel session—Security, privacy and the information processing system

SESSION CHAIRMAN—KENNETH ORR

Langston-Kitch and Assoc.

Panel Members

Jerry Hammett—Department of Defense
Daniel B. MaGraw—Department of Administration
Andrew O. Atkinson—PROJECT CLEAR—Regional Computation Center
Mark Gitenstein—Council Subcommittee on Constitutional Rights, U.S. Senate

OVERVIEW

With the publishing of the report of the Health, Education and Welfare Secretary's Committee, "Records, Computers and Rights of Citizens," on personal data systems the question of the individual's right to privacy and methods of securing this right have taken on even greater importance to the information processing committee. The panelists will attempt to define security and privacy and to draw the distinction which makes them both severable and related. In the past it has been a contention that security of systems is a panacea to the protection of privacy. It is the contention of some of the panelists that this is not necessarily the correct assumption.

A panel session—Venture capital for computer companies

SESSION CHAIRMAN—ROBERT F. JOHNSTON

Johnston Associates

Panel Members

John Doede—First Chicago Investment Corporation
Edgar Jannotta—William Blair & Company
Gene Amdahl—Amdahl Corporation
Moderator
Robert F. Johnston—Johnston Associates

OVERVIEW

This session will explain the factors investors take into consideration in evaluating venture capital investment. Investors are primarily interested in the management of the company, their previous performance, and their program for implementing their current business plan.

Conversely, we will also discuss the characteristics the management should look for in their investor. The investors should be experienced in venture capital, be aware of the risks and willing to work with the company.

We will also discuss the different types of financings—private placement and public offerings. The dramatic ups and downs of the stock market over the past several years have left the management of many small computer companies thoroughly confused regarding the criteria for investment. We hope to eliminate some of this confusion.

A panel session—The auditor/EDP manager relationship

SESSION CHAIRMAN—NOEL ZAKIN

AICPA

Panel Members

Richard J. Guiltinan—Arthur Anderson & Co.
Everett C. Johnson—Haskins & Sells
Richard D. Webb—Touche Ross & Co.

OVERVIEW

The auditor and EDP operating personnel have many areas of mutual interest. More effective communication will enable both auditors and data processing professionals to better discharge their respective responsibilities. The auditor's responsibilities, his role in an EDP environment, the need for mutual cooperation and planning, the impact of advanced systems and the use of audit software will be among the topics discussed. The AICPA's responsibilities and activities in the EDP area will also be considered.

A panel session—Hardware and software concerns relating to industrial processes

SESSION CHAIRMAN—HENRY R. KOEN

OVERVIEW

Software development is the major concern in the delivery of industrial process control. This session will examine key guidelines in hardware and software that effect success. Six papers will be presented in a workshop atmosphere with statements likely to draw fire. This blue ribbon panel of speakers will then be required to defend their theory and discovery. High attendee interaction is anticipated.

System software for real time systems and the unique control and application programs necessary to satisfy user requirements are examined. Typical industrial data acquisition and control systems are presented to characterize control and application software for data collection, processing, storage and retrieval. The approach is segmented into areas of currently high activity.

DIGITAL CONTROL ALGORITHMS—DESIGN AND APPLICATION CONSIDERATIONS—CHARLES W. ROSS

Many of the same functional design considerations required for an analog controller are properly applied to digital controller design. This presentation examines requirements beyond normal controller functions such as proportional, integral, derivative, feedforward, and compensating control to also examine practical design considerations such as: initialization, bumpless transfer, antiwindup provisions, limiting conditions, man-machine interface, accuracy limitations, deadbands, bandwidth and range selection. This controller permits considerable flexibility for conventional applications and allows the selection of various control options. Examples of hierarchically interfacing the controller to other, possibly more sophisticated, levels are presented, including parameter and error adaption, model feedforward and feedback, and economic optimization.

PROGRAM LANGUAGE SELECTION—JOHN E. PEYTON, JR.

Programming languages express problems in terms familiar to the user and are also machine translatable. Commonly used languages have been widely accepted because they express wide ranges of problems in a few fundamental modes. But these modes are often cumbersome or inadequate for expressing process control problems. Several attempts have been made to overcome this but no widely accepted means has yet been developed. Selection of a programming language for process control involves thorough analysis of the characteristics of the intended application and familiarity with the characteristics of available languages. Several current efforts are attempting to identify fundamental characteristics required of process control language and means of language implementation, especially for minicomputers.

INPUT-OUTPUT SOFTWARE REQUIREMENTS FOR PROCESS CONTROL—RICHARD F. THOMAS, JR.

The designer of a process I/O system should be prepared to apply a variety of techniques in order to produce a system well suited to the needs of the application. At the level of acquisition of data one must consider the merits of maintaining data banks versus direct acquisition of data on demand from application programs. There are usually several different needs for identification of process signals and devices. Especially, the interests of process control applications and control system maintenance and diagnostics may be sufficiently different to justify two levels of interface to process I/O devices. Generally at the highest level interface one would prefer to refer to signals by name, or perhaps logical unit number, regardless of the physical process involved in acquiring or sending data. The definition of these software interfaces and the transformations of both references and data which are associated with them can have an important impact on both the efficiency and utility of a control system.

MICRO COMPUTERS IN INDUSTRIAL PROCESSES—RONALD P. ROBERTS

For this new and booming technology, terminology must be clarified before we draw analogies on the evolution of small and very small computers. Some must even be considered extremely small.

The MICRO PROCESSOR and LSI computer in systems and industrial process control applications are first treated from a technical viewpoint, then in terms of economics and the role of manufacturers, OEM's and end-users. Likewise a treatise is to be opened on programming and software considerations. Questions will be raised on scope, responsibility, support and industry standards.

Thinking and discussion will be stimulated with comments on the future; markets, applications proliferation of manufacturers: Pitfalls and Advantages.

DATA HIGHWAYS—DALE W. ZOBRIST

Data highways are used to connect a sundry of input devices with computers. For industrial control applications these highways must often handle large volumes of data over considerable distances. Serial data path technology developed during the past decade is being applied to many industrial control systems today. In addition to the obvious wire and installation cost reductions, serial data highways can help reduce the risks often associated with large industrial installations. Various parallel and serial highways will be discussed, along with existing and proposed standards for such highways.

MAN/MACHINE CONSIDERATIONS IN FUTURE PROCESS CONTROL SYSTEMS—RENZO DALLIMONTI

Modern computer and display technology already provide the tools for implementing bold and exciting innovations to the man/machine interfaces of the future. CRT displays and computer systems have now reached a stage where they can replace the traditional panel with its 30 to 100 feet of instrumentation. The old dream of a "control-room-on-a desk" is now feasible for large process units.

The obstacles to its widespread adoption will be neither cost nor technology. The constraints will be our understanding of the operator's job, his operating procedures, and our knowledge of human factors engineering!

A panel session—Applying computers in the research and development laboratory

MODERATOR—DR. JOHN R. KOSOROK
Battelle-Northwest

Panel Members

Russell L. Heath—Aerojet Nuclear Company
Theodore H. Kehl—University of Washington
Edward A. Kramer—Digital Equipment Corporation
Robert E. Mahan—Battelle-Northwest

APPLYING COMPUTERS IN THE RESEARCH AND DEVELOPMENT LABORATORY—DR. JOHN R. KOSOROK

To bring about a better understanding of the needs of laboratory researchers employing small computers and the potential of these computers for meeting such needs, a panel of two computer users, a system designer and a computer manufacturer representative will address the problem. The panel will discuss:

(1) electronic and electromechanical devices comprising a computer system (hardware),
(2) computer programs (software), and
(3) connecting circuitry linking laboratory instruments and the computer (interface).

The panel will discuss the hardware, software and interfaces which are in existing laboratory installations and currently available from manufacturers. On-line, real-time systems in which the laboratory worker can interact with the computer system to modify its operation will be emphasized. Attendees with questions and problems are urged to participate during the question and answer period.

In the application of laboratory computers, the prospective user of the computer system takes the first step in the system design by specifying the required performance. Detailed computer knowledge is not necessary for this first step, but the prospective user should be aware of potential computer applications and the general performance capabilities so as not to unnecessarily restrict performance requirements. The system designer then integrates computer hardware, software and instrument interfaces to construct a system that will have sufficient capability to meet the user's research requirements. System costs and performance are directly related to the software available with the computer. Software for specific applications may be available from commercial sources or may be developed by the system designer.

APPLYING COMPUTERS IN RESEARCH AND DEVELOPMENT LABORATORIES—R. L. HEATH

Nuclear physics laboratories in this country have played a pioneering role in the development and application of real-time dedicated computer systems in the research laboratory. The complex nature of experiments which are conducted in this field have provided an ideal environment for the use of high speed digital processes for control of equipment, acquisition and storage of data, and real-time processing. Characteristics of such systems include extensive use of interactive devices, graphics, specialized programming languages and communications links between processor systems. The problems related to systems design, component selection, software development and system utilization will be illustrated by describing several experimental systems from design concept to end use. These will include both single purpose systems employing small mini-computers and a complete laboratory data acquisition and analysis system employing several processors and a multi-task operating system. The objective will be to provide a basis for stimulation of discussion on the options available to the potential user in the design and implementation of laboratory systems.

LABORATORY AUTOMATION IN A UNIVERSITY—T. H. KEHL

Our laboratory constructs digital devices both as independent and computer-interfaced. State logic methods (fast becoming industry standards) are used for all devices. Standardization of interfaces are performed in a rack which can contain up to 24 independent real-time clocks, optically-coupled external interrupts, clock-driven interval interrupts, power drivers, etc. Because of the extensive computer and stand-alone controller construction, a semi-automatic wire-wrap machine and computer-aided design software has been utilized. This equipment has, in turn, allowed us to construct our own mini-computers.

Our software, a central concern of laboratory automation, has the features of: (a) direct interrupt driven high-level statements, (b) development in FORTRAN on a large computer, (c) constants alteration from mini-computer console and, (d) monitor loading of multiple mini-computer programs.

THE MANUFACTURERS' ROLE IN APPLYING COMPUTERS IN THE RESEARCH DEVELOPMENT LABORATORY—EDWARD KRAMER

A mini-computer manufacturer plays a significant role and has a substantial obligation in determining the state of the technology for the application of computers in the R&D laboratory. A major supplier of computers to the R&D community is obligated to properly anticipate the needs of the scientific community and develop the products and services to satisfy those needs.

A typical computer company spends approximately 10 percent of its total revenues in the R&D. It continuously tries to anticipate the needs of its targeted market place and produce the products that are usable by its potential customers. The products today include not only hardware for data acquisition, computation and analysis, but also software and services that will allow these computer systems to be utilized more easily by the user community. The trade-off with cost, performance and reliability/maintainability serve as inputs in the development of new products. Consideration must also be given to any changes in the types of R&D users will be performing.

We will discuss how these factors interact to produce a product line that meets the needs of the scientific community. Other items to be discussed will include what can a user expect from a computer manufacturer, and general purpose tools versus dedicated applications.

THE METAMORPHOSIS IN LABORATORY COMPUTING—ROBERT E. MAHAN

Discussions of the emerging state of the art in laboratory computing are presented from the system designer's viewpoint. Topics of interest include hardware, software, interfacing, and standardization. The fragmentation of the historical minicomputer into three distinct classes, the supermini, the standard mini and the micromini, is discussed with emphasis on systems experience in a research and development laboratory environment. Available software support and limitations are discussed for each class of machine. Emphasis is placed on the capabilities of systems software; especially compilers, assemblers, and operating systems. Interface conventions, requirements, and experiences are discussed for several real-time systems. Of special interest is the treatment of the isolation and data conversion problem in the computer/instrument interface. The final topic presented concerns the standardization of system elements to promote the compatibility and reproduction of hardware and software in the laboratory environment. Benefits and limitations of standardization are cited.

A panel session—Communication nets in transportation

SESSION CHAIRMAN—DAN E. COUCHENOUR

Eastern Air Lines, Inc.

Panel Members

C. F. Norton—Northwest Orient Airlines
Donald LePorte—Transport Data Communications, Inc.
J. Swartz—Aeronautical Radio, Inc.
Moderator
Dan E. Couchenour—Eastern Air Lines, Inc.

OVERVIEW

Through development and application of combined computer and communications technology the transportation telecommunication network provides the optimum facility utilization. Standardization of data interchange formats, computer-to-computer interfaces, and network control procedures are today a reality. Topics will include: (1) Communication network planning and implementation, (2) Standardization of data interchange, (3) Network consolidation, (4) Computer-to-computer line control procedures used to provide assurance of data interchange, and (5) A panel discussion: Questions and Answers.

A panel session—Equipment control in transportation

SESSION CHAIRMAN—EUGENE JONES

Avis Rent-A-Car

OVERVIEW

At no time in the history of this industry has the control of its equipment been as vital. The energy crisis has resulted in a forced reduction of inventory. Control over the availability of this inventory is going to be the key not only to short term profits, but to providing the maximum level of customer service in this highly competitive industry which has significant long term implications.

In addition to the use of on-line reservation systems in the airline industry, current systems in other applications, such as the Wizard of Avis car rental system will be discussed.

A panel session—Computerized transportation-distribution—A user overview

SESSION CHAIRMAN—JACK W. FARRELL

Traffic Management Magazine

Panel Members

William R. McCartin—The Noxell Corporation
Frank J. Cyrkiewicz—Gerber Scientific Instrument Co.
Edmund R. Piesciuk—The Carrier Corporation
Phillip T. Catalano—Steelcase, Inc.
Moderator
Jack W. Farrell—Traffic Management Magazine

OVERVIEW

BUSINESS LOGISTICS is an ever-growing customer for computer services in manufacturing or marketing-oriented firms. Transportation, inventory control, order processing and myriad related activities require intensive computer support to meeting competitive pressures, reduce operating costs and improve customer services.

At this meeting, a physical distribution journalist, an MIS manager and three traffic-distribution executives discuss logistics development's urgent need for greater EDP-management participation.

A panel session—Computing and mathematics in society

MODERATOR—DR. DONALD L. THOMSEN, JR.
SIAM Institute for Mathematics and Society

Panel Members

Dr. Warren J. Ewens—University of Pennsylvania
Dr. James S. Coleman—University of Chicago
Dr. John J. Donovan—Massachusetts Institute of Technology
Dr. Warren E. Walker, The—New York City Rand Institute

PANEL OVERVIEW—DONALD L. THOMSEN, JR.

The Panel will address selected areas of current interest where computing and mathematics have been successfully applied to societal problems. Those societal fields which have been chosen are biological systems, sociology, energy, and urban emergency services. Although interrelated each area has quite different characteristics. In biology we examine long term evolutionary phenomena; in sociology we review through the techniques of simulation and surveys those human relations which are of concern day by day, month by month, and year by year; in energy we discuss problems both short and long range having to do with basic sources and distribution; and in emergency urban services we address very immediate activity related to what happens when that fire alarm box is pulled down the street. In these four areas the Panel will present examples of noteworthy past successes and conjectures as to where future progress very likely will be made.

COMPUTERS IN REAL-WORLD BIOLOGICAL SYSTEMS—WARREN J. EWENS

Real-world biological systems are both complex and long-lasting. The complexity arises through the fact that various characters are influenced by many genes, and also by the environment, that some genes influence many characters, and that the effects of the various genes are strongly interactive. To carry out programs of plant and animal breeding, as well as to study long-term evolutionary effects, only by using a computer can we hope to model these systems faithfully and draw useful conclusions.

COMPUTERS IN THE STUDY OF SOCIAL PROCESSES—JAMES S. COLEMAN

Two major directions in the use of computers exist in sociological work. One is the use in analysis of social surveys and social experiments, ranging from small to massive, and involving both data processing and statistical calculations. The second case is simulation of social processes, similar to, but with some variations upon, simulation in other areas. Examples of both these kinds of uses will be presented.

MANAGEMENT INFORMATION SYSTEMS FOR ENERGY DISTRIBUTION—JOHN J. DONOVAN

The problems facing us now are maldistribution, poor strategies, and in some cases a surplus of energy fuel. Our objective is to develop an energy information system that will provide management assistance in administering the distribution of energy resources. Functions of the system range from straight information retrieval to construction of the mathematical model for forecasting supply and demand and to construction of modeling facilities for testing the effects of various policies. Research in the construction of information systems, system privacy and security, and in file system design are particularly applicable. Using this research it has been possible to build such a system for the New England Region in a matter of months.

MATHEMATICAL MODELLING OF URBAN EMERGENCY SERVICE VEHICLE DEPLOYMENT POLICIES—WARREN E. WALKER

A sharp increase in fire alarms and a tightening of the expense budget in New York City resulted in a need for more effective use of the existing resources of the Fire Department. A computer simulation of the Department's operations and other mathematical models were developed to evaluate alternative deployment policies. Analysis which led to the implementation of new deployment policies will be discussed.

DATA REFERENCE CHARACTERISTICS OF DATABASE APPLICATION PROGRAMS

by ISAO MIYAMOTO

Nippon Electric Company Ltd.
Fuchu, Tokyo, Japan

ABSTRACT

In this paper the data reference characteristics of database applications programs are discussed. Factors characterizing some properties of data references are selected, and the data working set as a locality and the interreference interval distribution are measured.

The data reference characteristics of application programs can be divided into three types, hard-type, soft-type and special-type, when the logical structures of database and the search strategies are considered.

Also the problem of restructuring an application program taking parallelism into consideration is discussed.

MULTI-CONNECTION NETWORKS

by GERALD M. MASSON

University of Pittsburgh
Pittsburgh, Pennsylvania

ABSTRACT

Many applications involving the manipulation of large amounts of data or the interconnection of various subsystems into specific configurations employ the use of connection networks. Accordingly, a number of impressive results have been developed by the researchers in this field concerning the analysis and design of such networks. However, the vast majority of this work has dealt almost exclusively with a class of connection requirements for which at any time each input to the network is required to have at most one connection to the outputs of the network. It is becoming increasingly evident that there are a significant number of important applications for which this one-to-one type of connecting capability is not sufficient and for which a one-to-many or many-to-many type of connecting capability is needed instead. Networks capable of handling such requirements will be referred to as multi-connection networks, and this paper will consider their analysis and design. Aspects of the extension of existing work concerning the one-to-one type networks will be discussed. In addition, an approach will be described to the design of such networks which results in a new class of structures having multiple connection capability.

STAREX—THE JPL-STAR COMPUTER RESIDENT EXECUTIVE

by JOHN A. ROHR

Jet Propulsion Laboratory
Pasadena, California

ABSTRACT

STAREX, the resident executive for the JPL-STAR computer is presented. The JPL-STAR computer is a fault-tolerant computer which includes hardware self-repair capability. All programs which run on the computer require special considerations for proper operation in the fault-tolerant hardware environment. System programs use a single variable to indicate the rollback status of the active routine. User programs periodically invoke system routines to save the program state and establish rollback points. When a fault occurs, hardware self-repair occurs first, followed by software recovery to resume computation in a proper state. A brief description of the JPL-STAR computer is given, followed by an overview and then implementation details of STAREX. Finally, the fault-tolerance considerations in the design of STAREX are discussed.

SIMULATION OF EXECUTING ROBOTS IN IMPERFECTLY KNOWN ENVIRONMENTS*

by L. SIKLOSSY and J. DREUSSI

University of Texas
Austin, Texas

ABSTRACT

A simulated robot solves tasks in environments which she knows only approximately. The robot is given a description of the environments and of her capabilities. From the latter, she generates procedures that are evaluated to solve tasks. As tasks are solved, the robot improves her knowledge of the environment and the efficiency with which she can solve problems. Unknown, correctly known, and incorrectly known facts are treated in a uniform manner.

The design used, that of an *executing* robot, is contrasted to the design of *planning* robots. Executing robots may also be used to plan in perfectly known environments and are usually more efficient than planning robots. In imperfectly known environments, planning robots are inadequate.

* Partially supported by N.S.F. Grant GJ-34736.

THE PROVISION AND USE OF ENVIRONMENTAL INFORMATION IN A MULTIPROGRAMMING SYSTEM*

by TOMLINSON G. RAUSCHER

University of Maryland
College Park, Maryland

ABSTRACT

In a modern multiprogramming system a user, who is competing for system resources, may desire to optimize himself relative to his environment. The environment includes not only the physical machine and attached devices but also system software and the programs of other users which are sharing and competing for resources. As most modern systems provide little environmental information to users, we describe facilities for querying the system to ascertain environmental information. The user, in dynamically querying the environment, can use information on the status of hardware and software (both system and other user) to improve the status and performance of his program according to some cost function he selects. Several examples demonstrate the utility of these environmental inquiry facilities.

* This research was supported in part by the United States Atomic Energy Commission through Contract AT-(40-1)-3817.

BILL OF MATERIAL AND REQUIREMENTS CALCULATION SYSTEMS

J. WEINBERG
McDonnell Aircraft Company

ABSTRACT

The Bill of Material and Requirements Calculation System to be presented comprises the first phase of a total part number data base development that is expected to encompass all areas of production and inventory control within the McDonnell Aircraft Company component of McDonnell Douglas Corporation.

The system develops a bill of material file that contains both an engineering product configuration and a manufacturing product configuration. The file is created and maintained using both on-line real time and batch data entry methods. Included in the applications processed against the bill of material are programs which: perform a configuration audit to insure that the engineering and manufacturing configurations agree; calculate requirements for all manufactured parts; perform bill of material breakdowns and generate product structure listings; and perform document follow-up and status reporting in manufacturing operational areas.

The data base has been developed using the data management and teleprocessing facilities provided by the Information Management System (IMS). On-line update and retrieval is accomplished via a network of cathode ray tubes, on-line typewriters and on-line medium speed line printers located in operational areas.

1974 NATIONAL COMPUTER CONFERENCE COMMITTEES

General Chairman

Stephen S. Yau
Northwestern University
Evanston, Illinois

Vice Chairman

Samuel Levine
United Air Lines
Chicago, Illinois

Computer Art Fair

Kurt F. Lauchner
Eastern Michigan University
Ypsilanti, Michigan

Computer Science Fair

Benjamin Mittman
Northwestern University
Evanston, Illinois

Science Theatre

Thomas Murphy
University of Chicago
Chicago, Illinois

Tours

John Mayer and Judy Newman
United Air Lines
Chicago, Illinois

Controller and Secretary

James S. Aagaard
Northwestern University
Evanston, Illinois

NCCC Representative

Albert K. Hawkes
Computer Horizons, Inc.
Chicago, Illinois

Technical Program Committee

Theodore M. Bellan—*Chairman*
McDonnell Douglas Automation Co.
St. Louis, Missouri

C. V. Ramamoorthy
University of California
Berkeley, California

R. Stockton Gaines
Institute of Defense Analyses
Princeton, New Jersey

Thomas N. Pyke, Jr.
National Bureau of Standards
Washington, D. C.

Herbert Seidensticker
Combustion Engineering, Inc.
Stamford, Connecticut

James A. Schweitzer
Xerox Corporation
Rochester, New York

W. Chou
Network Analysis Corp.
Glen Cove, New York

Walter S. Huff, Jr.
Huff, Barrington and Owens and Associates

Erik D. McWilliams
National Science Foundation
Washington, D. C.

Thomas J. Archbold
International Harvester Company
Hinsdale, Illinois

Vernon L. Schatz
The Jewel Company, Inc.
Chicago, Illinois

Gerald T. Montgomery
J. C. Penney Company
New York, N. Y.

Vern H. Tanner, Jr.
State of Iowa
Des Moines, Iowa

Allen J. Burris
The Northern Trust Company
Chicago, Illinois

William D. Tabachnik
Mobil Oil Corporation
New York, New York

Paul G. Mercer
Eastern Airlines
Miami, Florida

Local Arrangements Committee

Richard B. Wise—*Chairman*
IIT Research Institute
Chicago, Illinois

James M. Francoeur
IIT Research Institute
Chicago, Illinois

Robert S. Hollitch
Sargent & Lundy Engineers
Chicago, Illinois

DISCUSSANTS, MODERATORS AND PANELISTS

Abrams, Marshall
Alsburg, Peter
Amdahl, Gene
Anastatio, E. J.
Anderson, John W.
Ando, Kaoru
Appleton, Jon
Atkinson, Andrew O.
Ayres, M. U.

Bachman, C. W.
Bacon, Glen
Baker, Laurence H.
Ball, Marion J.
Battiste, Edward
Bell, Thomas E.
Benes, V. E.
Bialek, Fred
Bigelow, Robert P.
Bjorklund, Howard
Blanc, Robert
Blanchard, Morton
Blasch, Larry
Blazer, Robert M.
Boase, Ronald L.
Bockelman, Melvin F.
Boehm, Barry W.
Bouknight, W. Jack
Brebach, Gresham T.
Brennan, Gregory L.
Brookings, Doug
Brooks, F. P. Jr.
Brown, N. H.
Burgess, P. M.
Butler, Jack

Canada, Ralph
Caravella, Robert T.
Carey, Charles G.
Carey, W. E.
Carruth, Ronald
Casali, Harold O.
Catalano, Phillip T.
Cerf, Vinton
Cherry, Herbert
Cody, William J.
Coleman, James S.
Conners, R. R.
Cornelius, John
Crandall, Richard L.
Crocker, Stephen R.
Cross, Phillip C.
Cyrkiewicz, Frank J.

Dallimonti, Renzo
Denicoff, Marvin
Dertouzos, M. L.
Dewar, Robert B. K.
Doede, John
Donohue, B. P. II
Donovan, John
Donovan, John J.
Doyle, W. S.
Drew, Adrian
Dunten, Stanley D.

Elchesen, Dennis
Epich, Raymond
Estrin, Gerald
Ewens, Warren J.
Farmer, Robert S.
Feustel, Edward
Fiala, Edward
Finerman, Aaron
Firestone, Roger M.
Fisher, Charles R.
Foley, James D.
Fossum, Barbara
Freeman, R. D.
Fried, Richard
Friedman, Richard
Friel, William

Gardner, Willard
Garland, Stephen J.
Garratt, Alfred H.
Gear, William
Gibbons, J. R.
Giebink, Gerald A.
Gimpel, James F.
Gitenstein, Mark
Glacken, Tom
Glenn, William V. Jr.
Godwin, William
Goldstine, Hermann
Goodyear, Franklin F.
Gotlieb, C. C.
Grad, Burton
Grishman, Ralph
Grosch, H. R. J.
Grossman, Jerome H.
Guiltinan, Richard J.
Gustafson, David

Hagerty, P. E.
Haggerty, J. P.
Hamilton, Alan A.
Hammer, Carl
Hammett, Jerry
Hanson, David R.
Hargraves, Robert F.
Harrington, Rodney B.
Harris, Larry R.
Harrison, R. D. Jr.
Harvey, Samuel
Heath, Russell L.
Hilton, Robert
Hoff, Marcian E. Jr.
Hoffman, Lance J.
Holliday, Roger
Hollister, James N.
Hopper, Grace M.
Horne, William J.
Howard, Phillip C.
Howell, Nelson A.
Hughes, Arthur D.
Hunter, Kenneth W.
Hsiao, David

Ineson, John

Jackson, H. M. II
Jannotta, Edgar
Jardine, Don
Johnson, Everett C.
Johnson, Robert
Johnson, Robert L.
Johnson, Whitney L.

Kaellner, C. G.
Kandel, Abraham
Kehl, Theodore H.
Kemeny, John G.
Kilgour, Frederick G.
King, Bernard F. Jr.
King, John
Kiviat, Philip J.
Klink, William C.
Koenig, Robert
Kramer, Edward A.
Kulikowski, Casimir
Kuo, Frank

Lanahan, John R.
Landim, E.
Lawson, Charles
LePorte, Donald
Linden, Theodore A.
London, Phillip J.
Long, Harvey
Low, Ron
Lowenthal, E.
Luck, Dennis
Luehrmann, Arthur W.
Luke, John W.
Lusted, Lee B.

Macdonald, John B.
Machover, Carl

MacKinnon, Don
MacGraw, Daniel B.
Mahan, Robert E.
Mairet, Chick
Manion, Robert
Marsh, H. G.
McCarn, Davis B.
McCartin, William R.
McCluskey, Edward J.
McConnell, Tom
McDonald, J. F.
McGirt, Frank
McGovern, Patrick
McHale, Michael
McIlroy, M. D.
McManamon, Peter M.
Metcalfe, Robert
Milano, James V.
Mills, Richard D.
Minter, Dan
Mitchell, Victor
Montross, P. M.
Morgan, M. Granger
Mueller, Gerhard O.
Muenz, James
Munyan, John
Murphy, Daniel

Naughton, Joseph D.
Negroponte, Nichalos P.
Neumann, Peter G.
Nievergelt, Jurg
Norton, C. F.

O'Connell, Mike
Olson, J. W.
Ornstein, Severo
O'Rourke, Thomas
Oser, Hans J.
Ostlund, James J.

Parker, Donn G.
Paulson, Lee
Peleyras, Francois
Perra, John
Peters, Carol
Peyton, John E. Jr.
Philipps, Lou
Piesciuk, Edmund R.
Pinkerton, Tad B.
Pinson, Elliot N.
Pirtle, Mel
Popek, Gerald J.
Postel, Jonathan B.
Pouzin, Louis

Rasmussen, Norman
Reinstedt, Robert
Renninger, Clark R.
Reynolds, Carl H.
Richardson, Duane
Richley, T.
Roberts, Ronald P.
Rosenthal, C. W.
Ross, Charles W.
Ruth, Stephen
Ryniker, Richard

Salbu, Eric
Samek, Michael J.
Santos, Paul J.
Scanlon, Robert
Schell, Roger R.
Schlafly, Hubert J.
Scott, Ben
Scott, Robert H.
Screenivasan, K.
Sears, Dennis
Shaffer, Charles
Sheer, Sol
Short, Gerald E.
Siegel, John H.
Simon, Richard
Skinner, Friend
Smythe, Sheila
Soden, John V.
Sprague, Richard
Steel, T. B. Jr.
Stibitz, George
Stotz, Robert H.
Strasburg, Harry
Swanson, Mary Ann
Swartz, J.

Tiechroew, D.
Thomas, Richard F. Jr.
Thomas, Robert
Thornton, Zane
Tomlin, M. G.
Trainor, James J.
Trigg, Charles D.
Tucker, Dorothy I.
Turn, Rein
Tutelman, David
Tyler, S. S.

Uhlig, Ronald P.
Ullman, Art
Uretsky, Myron

Vale, Joseph
Villalobos, Luis

Walden, David
Walker, Warren E.
Ward, James A.
Warms, Jon
Webb, Richard D.
Weingarten, Frederick W.
Weissman, Clark
Welke, L. A.
White, Henry J.
White, Terry
Williams, E. Belvin
Williams, Leland H.
Wood, Bryan
Wulf, William
Wulf, William A.
Wyatt, Joseph

Yabobbin, Ray
Yamada, Gordon

Zemanek, H.
Zobrist, Dale W.

REVIEWERS

COMMUNICATIONS SYSTEMS

Babayan, V.
Bender, E. C.
Byrne, E. R.
Carkin, P.
Conners, R. R.
Desoer, C. A.
Gearing, B.
Giloth, P. K.
Gordon, T. H.
Herbst, R. T.
Huttenhoff, J.
Janik, J. J.
Koeppen, C.
Mansell, J. J.
Nissley, M.
Ossanna, J. F., Jr.
Pederson, D. O.
Pehlert, W. K.
Peterson, T. G.
Reines, J.
Riebe, R.
Rochkind, M. M.
Scanlon, J. M.
Shoaf, G. H.
Slana, M. F.
Tuomenoksa, L. S.
Whitehead, L. D.

COMPUTER ARCHITECTURE & HARDWARE

Baer, J.
Belady, L. A.
Compaigne, H.
Carter, W. C.
Chandy, K. M.
Chen, T. C.
Chu, W. W.
Chu, Y.
Cragon, H. G.
Davidson, C. H.
Feng, T.
Foster, C. C.
Garcia, O. N.
Harlow, C. A.
Hong, S. J.
Kruy, J. F.
Kuck, D. J.
Manning, E. G.
Mathur, F.
McCluskey, E. J.
Merwin, R. E.
Mulder, M. C.
Parish, R. M.
Sassenfeld, H. M.
Siewiorek, D. P.
Simmons, R. B.
Stone, H. S.
Szygenda, S. A.
Trauboth, H. H.
Walford, R. B.
Watson, W. J.
Yeh, R. T.

COMPUTER NETWORKING

Alter, R.
Ashenhurst, R. L.
Aupperle, E. M.
Blanc, R. P.
Bouknight, J. W.
Bowles, K.
Cotton, I. W.
Farber, D.
Frank, H.
Jasper, D. P.
Kleinrock, L.
Kuo, F. K.
Metcalfe, R.
Wulf, W.

DISTRIBUTION

Bonner, W.
Koenig, R.
Paulson, L.
Wiorkowski, G.

EDUCATION

Ashenhurst, R.
Bitzer, D.
Boast, W.
Booth, T.
Brooks, F.
Collins, G.
Glaser, R.
Harley, W.
Hitchens, H.
Kerr, E. G.
Mills, H.
Morgan, G.
Myers, R.
Suppes, P.
Tiedeman, D.
Wedemeyer, C.

HEALTH CARE & BIOTECHNOLOGY

Crean, D. M.
Dunn, M. D.
Ganter, G. E.
Jacob, S. E.
Korn, H. E.
Macaleer, R. J.
McGrath, E. T.
Pickens, K. E.
Wasserman, A. I.
Zwicky, G. F.

INDUSTRIAL PROCESS CONTROL

Allan, J. J.
Bergen, W. S.
Corripio, A. B.
Fisher, J. M.
Johnson, A. E., Jr.
Koppel, L. B.
Lavigne, J. R.
Reduto, E. P.
Scioscia, J. S.
Stenuf, T. J.
Westerberg, A. W.
Whitman, K. A.
Whooley, J. P.

INFORMATION MANAGEMENT SYSTEMS

Baledes, T.
Blackmare, S.
Bonekamp, F.
Canning, R. C.
Case, L. R.
Codd, E. F.
Fecenko, T.
Gumerman, A. F.
Hernon, J.
Joy, J.
Katter, D. A.
Kirshenbaum, F.
Little, J.
Liver, J. A.
Lupien, C.
Mairet, C. E.
Michey, M.
Murray, E.
Myers, W.
O'Neil, N. D.
O'Toole, J.
Parke, B.
Pattern, F.
Prescott, L.
Rittersbach, G. H.
Schoen, A.
Secrest, R.
Smith, L. B.
Stanford, D.
Tilton, F.
Vaughn, P.
Vreugdenhill, C. K.
Weatherman, J.
Wichael, D.
Zakrzewski, R. S.

MANAGEMENT ACCEPTANCE

Aron, J.
Bateman, B.
Carlson, G.
Cheek, R.
DeVine, V.
Dooley, R.
Gilchrist, B.
Horowitz, P.
Krause, K. W.
Matye, T.
Perry, R.
Schlegel, M.
Wofsie, M.

SOFTWARE SYSTEMS

Ball, N. A.
Branstad, D. K.
Buzen, J. P.
Eckhouse, R. H., Jr.
Fabry, R. S.
Fife, D.
Gimple, J. F.
Glick, M.
Griswold, R. E.
Halstead, M. H.
Hodges, R.
Hoffman, L. J.
Hsiao, D. K.
Johnson, D.
Keller, R. M.
Lampson, B. W.
Lee, J. A. N.
Linden, T.
Liskov, B. H.
Lucas, B.
Lynch, W. C.
Lyon, G.
Morris, J. B.
Plauger, P. J.
Popek, G. J.
Roth, P.
Schlegel, C. T.
Schmitt, S. A.
Schroeder, M. D.
Sevcik, K.
Squires, S.
Stewart, S. L.
Stillman, R.
Varian, L.
Waite, W. M.
Walker, J.
Wexelblat, R.

SESSION CHAIRMAN AND AREA DIRECTORS

Alter, Ralph
Archbold, Thomas J.
Archibald, Julius A. Jr.
Ashenhurst, Robert L.

Bakalar, Thomas A.
Bateman, Barry L.
Berra, P. Bruce
Buchik, G.
Burchfiel, Jerry D.
Burris, Allen J.
Burrows, James H.

Cashman, Thomas J.
Charp, Sylvia
Chou, Wushow
Chu, Wesley W.
Clive, Jonathan
Cotton, Ira W.
Couchenour, Dan E.
Culbertson, Don S.
Cupp, B. Garland

Damon, Phillip P.
Danner, Lee
Davis, John C.
Denning, Peter J.
Donavon, Paul F.
Dunn, Michael D.
Durand, Gerald C.

Evensen, Don
Fabisch, M. P.
Farber, David J.
Farrell, Jack W.
Feeney, George J.
Frank, Howard

Gagliardi, Ugo O.
Gaines, R. Stockton
Galey, Michael
Gantner, George E.
Gentile, John
Goetz, Martin A.
Gomersal, Earl

Gonzalez, Mario J. Jr.
Griswold, Ralph E.

Harris, Fred H.
Harris, JoAnn
Haynes, Richard
Hobbs, L. Charlie
Huff, Walter S.
Hyduk, S. J.

Jacobs, Stanley E.
Jasper, David P.
Johnston, Robert F.
Jones, Eugene

Kahn, Bob
Kerr, Eugene C.
Kleinrock, Leonard
Koen, Henry R.
Kosorok, John R.
Kruy, Joseph
Kulsrud, Helene E.
Kurtz, Thomas E.

Lagabun, Gene
Lev, Joseph A.
Levine, Sam
Lipner, Steven B.
Lockett, JoAnn
Lupien, Charles A.
Lykos, Peter

Masson, Gerald M.
Mathis, Robert F.
Maurer, W. D.
McCormick, Bruce
McWilliams, Erik D.
Mercer, Paul G.
Merwin, Richard E.
Mitchell, George
Montgomery, Gerald T.

Nerad, Richard A.
Nolan, Richard L.

Oberg, James E.
Oliver, Paul

Orr, Kenneth
O'Toole, J.
Oyer, Paul D.

Patterson, William
Pommerening, Glen E.
Pyke, Thomas N.

Rahimi, Morteza A.
Ramamoorthy, C. V.
Raub, William F.
Rice, John R.
Richards, Murray L.
Richardson, William M.
Ridgeway, John R.
Rosen, Saul
Rosler, Lawrence

Schaatz, Vernon L.
Schreiber, H.
Schueitzer, James A.
Secrest, R. D.
Seidensticker, Herbert
Shaw, Jack
Simmons, Dick B.
Simonette, Larry
Solomon, I. I.
Stefferud, Einar
Stitelman, Leonard
Su, Stanley Y. W.
Tabachnik, William D.
Tanaka, Richard I.
Tanner, Verne H. Jr.
Thomsen, Donald L.
Thayer, Richard H.

Utlaut, William F.

Ware, Willis H.
Warner, James
Wasserman, Anthony I.
Wofsey, Marvin M.
Woodbury, Max A.

Zadeh, Lotfi A.
Zakin, Noel

AUTHOR INDEX