AFIPS CONFERENCE PROCEEDINGS

VOLUME 34

1969 SPRING JOINT COMPUTER CONFERENCE

May 14-16, 1969
Boston, Massachusetts

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

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

1969 SPRING JOINT COMPUTER CONFERENCE COMMITTEE

CONTENTS

A modular approach to file system design*

by STUART E. MADNICK

Massachusetts Institute of Technology
Cambridge, Massachusetts

and

JOSEPH W. ALSOP, II

International Computation Incorporated
Cambridge, Massachusetts

INTRODUCTION

A generalized model or "blue-print" for the design of sophisticated file systems is presented in this paper. The model exploits the concepts of "hierarchical modularity" and "virtual memory."

Any general file system design model must, of course, be modified and refined to satisfy the requirements of a specific environment. The details of the file system model are presented in three steps: (1) the basic concepts and overview are discussed, (2) an example environment consisting of a multi-computer network with the added complexities of coordination, structured file directories, and removable volumes is described, and (3) each of the hierarchical levels of the file system is elaborated in terms of the assumed environment.

Basic concepts used in file system design

Two concepts are basic to the general file system model to be introduced. These concepts have been described by the terms "hierarchical modularity" and "virtual memory." They will be discussed briefly below.

Hierarchical modularity

The term "modularity" means many different things to different people. In the context of this paper we will be concerned with an organization similar to that proposed by Dijkstra[6,7], and Randell.[14] The important aspect of this organization is that all activities are divided into sequential processes. A hierarchical structure of these sequential processes results in a level or ring organization wherein each level only communicates with its immediately superior and inferior levels.

The notions of "levels of abstraction" or "hierarchical modularity" can best be presented briefly by an example. Consider an aeronautical engineer using a matrix inversion package to solve space flight problems. At his level of abstraction, the computer is viewed as a matrix inverter that accepts the matrix and control information as input and provides the inverted matrix as output. The application programmer who wrote the matrix inversion package need not have had any knowledge of its intended usage (superior levels of abstraction). He might view the computer as a "FORTRAN machine", for example, at his level of abstraction. He need not have any specific knowledge of the internal operation of the FORTRAN system (inferior level of abstraction), but only of the way in which he can interact with it. Finally, the FORTRAN compiler implementer operates at a different (lower) level of abstraction. In the above example the interaction between the 3 levels of abstraction is static since after the matrix inversion program is completed, the engineer need not interact, even indirectly, with the applications programmer or compiler implementer. In the form of hierarchical modularity used in the file system design model, the multi-level interaction is continual and basic to the file system operation.

* Work reported herein was supported (in part) by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

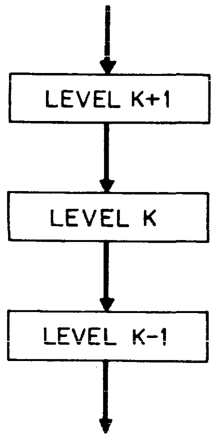


Figure 1—Hierarchical levels

There are several advantages to such a modular organization. Possibly the most important is the logical completeness of each level. It is easier for the system designers and implementers to understand the functions and interactions of each level and thus the entire system. This is often a very difficult problem in very complex file systems with tens or hundreds of thousands of instructions and hundreds of interdependent routines.

Another by-product of this structure is "debugging" assistance. For example, when an error occurs it can usually be localized at a level and identified easily. The complete verification (reliability checkout) of a file system is usually an impossible task since it would require tests using all possible data input and system requests. In order to construct a finite set of relevant tests, it is necessary to consider the internal structure of the mechanism to be tested. Therefore, an important goal is to design the internal structure so that at each level, the number of test cases is sufficiently small that they can all be tried without overlooking a situation. In practice, level 0 would be checked out and verified, then level 1, level 2, etc., each level being more powerful, but because of the abstractions introduced, the number of "special cases" remains within bounds.

Virtual memory

There are four very important and difficult file system objectives: (1) a flexible and versatile format, (2) as much of the mechanism as possible should be invisible, (3) a degree of machine and device independence, and (4) dynamic and automatic allocation of secondary storage. There have been several techniques developed to satisfy these objectives in an organized manner; the concept exploited in this generalized file system has been called "segmentation"[5] or "named virtual memory".[3] Under this system each file is treated as an ordered sequence of addressable elements, where each element is normally the same size unit as the main storage, a byte or word. Therefore, each individual file has the form of a "virtual" core memory, from whence the name of the technique came. The size of each file is allowed to be arbitrary and can dynamically grow and shrink. There is no explicit data format associated with the file; the basic operations of the file system move a specified number of elements between designated addresses in "real" memory and the "virtual" memory of the file system.

There are several reasons for choosing such a file concept. In some systems the similarity between files and main storage is used to establish a single mechanism that serves as both a file system for static data and program storage and a paging system[3,5,8] for dynamic storage management. "Virtual memory" provides a very flexible and versatile format. When specific formatting is desired, it can be accomplished by the outermost file system level or by the user program. For example, if a file is to be treated as a collection of card-image records, it is merely necessary to establish a routine to access 80 characters at a time starting at byte locations 0, 80, 160,. . . . Almost all other possible formats can be realized by similar procedures.

Except for the formatting modules, the entire file system mechanism, including allocations, buffering, and physical location, is completely hidden and invisible to the user. This relates closely to the objective of device independence. In many file systems the user must specify which device should be used, its record size (if it is a hardware formatable device), blocking and buffering factors, and sometimes even the physical addresses. Although the parameters and algorithms chosen might, in some sense, be optimal, many changes might be necessary if the program is required to run with a different configuration or environment.

There are very serious questions of efficiency raised by this file system strategy. Most of these fears can be eased by the following considerations. First, if a file is to be used very seldom as in program development, efficiency is not of paramount importance; if, on the other hand, it is for long-term use as in a commercial production program, the device-independence and flexibility for change and upkeep will be very important. Second, by relieving the programmer of the complexities of the formats, devices, and allocations, he is able to utilize his energy more constructively and creatively to develop clever algorithms relating to the

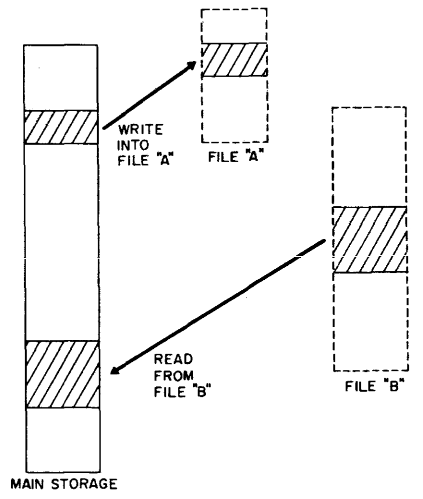


Figure 2—"Real" memory and "virtual" file memory

logical structuring of his problem rather than clever "tricks" to overcome the shortcomings or peculiarities of the file system. Third, in view of the complexity of current direct-access devices, it is quite possible that the file system will be better able to coordinate the files than the average user attempting to specify critical parameters.

Overview of file system design model

The file system design model to be presented in this paper can be viewed as a hierarchy of six levels. In a specific implementation certain levels may be further sub-divided or combined as required. A recent study of several modern file systems, which will be published in a separate report, attempts to analyze the systems in the framework of this basic model. In general all of the systems studied fit into the model, although certain levels in the model are occasionally reduced to trivial form or are incorporated into other parts of the operating system.

The six hierarchical levels are:

1. Input/Output Control System (IOCS)
2. Device Strategy Modules (DSM)
3. File Organization Strategy Modules (FOSM)
4. Basic File System (BFS)
5. Logical File System (LFS)
6. Access Methods and User Interface

The hierarchical organization can be described from the "top" down or from the "bottom" up. The file system would ordinarily be implemented by starting at the lowest level, the Input/Output Control System, and working up. It appears more meaningful, however, to present the file system organization starting at the most abstract level, the access routines, and removing the abstractions as the levels are "peeled away".

In the following presentation the terms "file name", "file identifier", and "file descriptor" will be introduced. Detailed explanations cannot be provided until later sections, the following analogy *may* be used for the reader's assistance. A person's name (file name), due to the somewhat haphazard process of assignment, is not necessarily unique or manageable for computer processing. A unique identifier (file identifier) is usually assigned to each person, such as a Social Security number. This identifier can then be used to locate efficiently the information (file descriptor) known about that person.

Access Methods (AM)

This level consists of the set of routines that superimpose a format on the file. In general there will probably be routines to simulate sequential fixed-

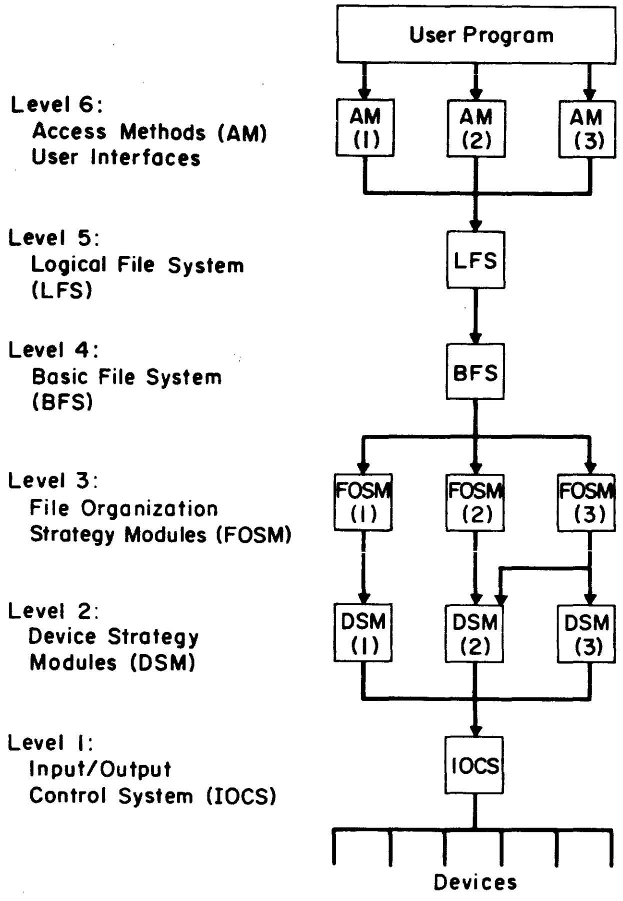


Figure 3—Hierarchical file systems

length record files, sequential variable-length record files, and direct-access fixed-length record files, for example. Many more elaborate and specialized format routines, also called access methods or data management, can be supplied as part of the file system. Obviously, a user may write his own access methods to augment this level.

Logical File System (LFS)

Routines above this level of abstraction associate a symbolic name with a file. It is the function of the Logical File System to use the symbolic file name to find the corresponding unique "file identifier". Below this level the symbolic file name abstraction is eliminated.

Table I—Example procedure to perform logical file system search

```
DECLARE    1 FILE_ENTRY,
           2 FILENAME CHARACTER (8),
           2 VOLUME   CHARACTER (8),
           2 INDEX    FIXED BINARY,
                 . . .,
                 . . .
DO    I = 1 TO PATH_LENGTH;
      DO    J = 0 BY N WHILE (FILE_ENTRY.FILENAME ¬= PATH(I));
            CALL BFS_READ(BASE_VOLUME,BASE_INDEX,FILE_ENTRY,J*N,N);
      END;
      BASE_VOLUME = FILE_ENTRY.VOLUME;
      BASE_INDEX = FILE_ENTRY.INDEX;
END;
                 . . .
                 . . .
```

Basic File System (BFS)

The Basic File System must convert the file identifier into a file descriptor. In an abstract sense, the file descriptor provides all information needed to physically locate the file, such as the "length" and "location" of the file. The file descriptor is also used to verify access rights (read-only, write-only, etc.), check read/write interlocks, and set up system-wide data bases. The Basic File System performs many of the functions ordinarily associated with "opening" or "closing" a file. Finally, based upon the file descriptor, the appropriate FOSM for the file is selected.

File Organization Strategy Modules (FOSM)

Direct-access devices physically do not resemble a virtual memory. A file must be split into many separate

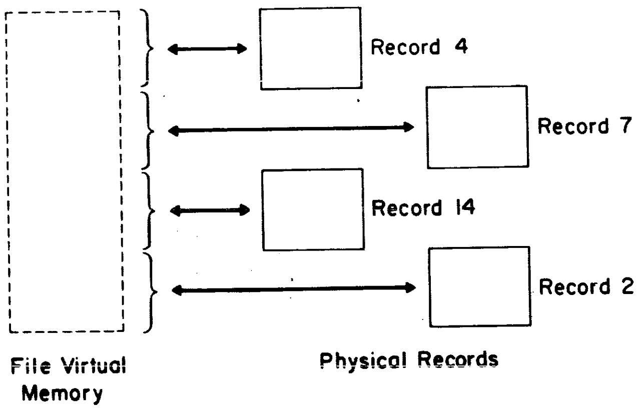


Figure 4—Mapping virtual memory into physical records

physical records. Each record has a unique address associated with it. The File Organization Strategy Module maps a logical virtual memory address into the corresponding physical record address and offset within the record.

To read or write a portion of a file, it is necessary for the FOSM to translate the logically contiguous virtual memory area into the correct collection of physical records or portion thereof. The list of records to be processed is passed on to the appropriate DSM.

To minimize redundant or unnecessary I/O, the FOSM allocates "hidden" file buffers as needed. If the requested portion of virtual memory is contained in a currently buffered record, the data can be transferred to the designated user main storage area without intervening I/O. Conversely output to the file may be buffered. If a sufficiently large number of buffer areas are allocated to a file, it is possible that all read and write requests can be performed by merely moving data in and out of the buffers. When a file is "closed", the buffers are emptied by updating the physical records on the secondary storage device and releasing them for use by other files. Buffers are only allocated to files that are actively in use (i.e., "open").

Device Strategy Modules (DSM)

When a large portion of a file is to be read or written, many records must be processed. The Device Strategy Module considers the device characteristics such as latency and access time to produce an optimal I/O sequence from the FOSM requests.

The DSM also keeps track of the available records on the device. It is responsible for allocating records for a file that is being created or expanded, and deallocating records for a file that is being erased or truncated. The FOSM requests that a record be allocated when needed, the DSM selects the record.

Input/Output Control System (IOCS)

The Input/Output Control System coordinates all physical I/O on the computer. Status of all outstanding I/O in process is maintained, new I/O requests are issued directly if the device and channel are available, otherwise the request is queued and automatically issued as soon as possible. Automatic error recovery is attempted when possible. Interrupts from devices and unrecoverable error conditions are directed to the appropriate routine. Almost all modern operating systems have an IOCS.

File systems versus data management systems

In the literature there is often confusion between systems as described above, which this paper calls "file systems" and systems which will be called "data management systems", such as DM-1,[8] GIM-1,[13] and TDMS.[17] The confusion is to be expected since both types of systems contain all of the functional levels described above. The systems differ primarily on the emphasis placed on certain levels.

In general file systems, the file is considered the most important item and emphasis is placed on the directory organization (Logical File System) and the lower hierarchical levels. It is expected that specialized access methods will be written by users or supplied with the system as needed.

In most data management systems, the individual data items are considered the most important aspect, therefore emphasis is placed on elaborate access methods with minimal emphasis on the lower levels of abstraction. Because of the heavy emphasis on a single level, data management systems tend to appear less hierarchical than file systems since the lower levels are often absorbed into the access methods.

Multi-computer network environment

A general file system design model must, of course, be modified and elaborated to satisfy the needs of any specific desired file system environment. To illustrate the refinement process, a unique file system design will be presented for a multi-computer network.

Multi-computer networks are becoming an increasingly important area of computer technology.[11] There are several significant reasons behind the growth of multi-computer networks:

1. To increase the power of a computer installation in a modular manner, especially if (a) it is not possible to acquire a larger processor, (b) reliability is important, or (c) there are real-time or time-sharing constraints.
2. To serve the coordination requirements of a network of regional computer centers.
3. To support the accessibility to a nation-wide data base.

An example of the environment to be considered for this paper can be illustrated in Figure 5. This type of multi-computer network has been in limited use for several years in many configurations. The IBM 7094/7044 Direct-Coupled System was probably one of the earliest practical examples of such an interconnected arrangement.

There are several implicit constraints imposed upon the multi-computer system illustated in Figure 5:

1. Independence of Central Processors.
   Each of the central processors operates independently such that there are no direct processor-to-processor data transfer nor signaling, and furthermore there is no "master" processor.
2. Non-shared Memory.
   Each central processor has its own main storage unit. These units are not shared with nor accessed by another central processor.
3. Inter-locked Device Controllers.
   The device controllers act as "traffic cops" to the actual I/O direct access devices. They control the traffic between a computer's I/O channel and a selected I/O device. A single device controller will only accept requests from one channel at a time and will only select one I/O device (among those under its control) at a time. Once a device controller connects a

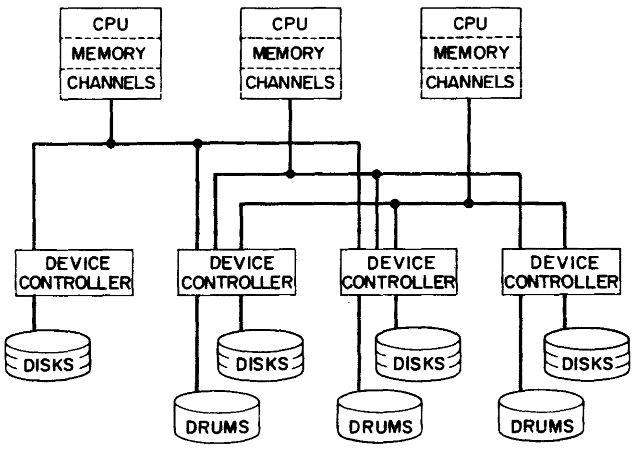


Figure 5—Example of multi-computer file system network

channel with a device, the connection remains intact until the channel releases the device or an I/O error occurs.

The environment described above, although well within the boundaries of current technology, has not been the subject of much investigation. Such configurations are presently very expensive and, therefore, chosen only for very specialized situations. Even then there are only two or three processors and very specialized software and operational factors. A discussion of the CP-67/CMS Time Sharing System [9,21] will serve to establish the relevance of the multi-computer network environment.

The CP-67/CMS Time Sharing System uses the special hardware features of a single IBM System/360 model 67 processor augmented by software to produce an apparent environment corresponding to the multi-computer network illustrated in Figure 5, with many independent central processors, device controllers, and direct access I/O devices. In practice a typical single processor 360/67 configuration would produce the affect of about 30 active processors ("virtual" System/360 model 65 processors each with a 256,000 byte memory) and 50 active device controllers. More detailed descriptions of the CP-67/CMS System can be found in the References. In the traditional sense of time-sharing, each user of the CP-67/CMS System is provided with a "virtual" computer operated from a simulated operator console (actually an augmented remote terminal). Most importantly, each "virtual" computer (i.e., user) operates logically independently of all other "virtual" computers except for the specified inter-connected I/O devices and device controllers.

Problems arising in multi-computer networks

There are many problems associated with the multi-computer file system network. Some of these problems are unique to this environment. Other problems have been solved in traditional file systems,[2,17,20] but the solutions require major revisions due to the peculiarities of the environment. The most significant problems are listed briefly below.

1. No shared memory.
   Usually file systems coordinate the status of the files and devices by using main storage accessible tables and data areas that describe file status, access rights, interlocks, and allocation. There is no such common communication area in main storage that can be accessed by all the independent processors.
2. No inter-computer communication.
   Multi-computer configurations usually provide a mechanism for sending signals or data transfers between the separate processors. With this capability the non-shared memory problem could be solved by either (a) electing one processor to be the "master" processor that coordinates the other processors, or (b) supply all the processors with enough information such that each processor knows what all the other processors are doing. The concept of a "master" processor opposes the intended homogeneous, independent processor assumption. The possibility of supplying status information to all other processors, although reasonable for a three or four processor configuration, was not considered a feasible solution for a system with hundreds of processors and devices and thousands of files. For these reasons, inter-computer communication, although an available capability, was not included as a required capability of the multi-computer environment described above.
3. No pre-arranged allocations.
   For small specialized multi-computer file networks, each processor can be "assigned" a specific area of a device or set of devices that can be used to write new files, all other processors can only read from this area by convention. This prevents the danger of two independent processors writing files at the same place. Such an "arrangement" is not practical for a large, flexible multi-computer file network since the static assignment of secondary storage space does not take account of the dynamic and unpredictable requirements of the independent processors.
4. Extendable device and file allocation.
   The number of devices and sizes of devices as well as the number and sizes of files are, within reason, unlimited. For example, a specific amount of secondary storage equivalent to 100,000 card images could be used to hold 10 files of 10,000 cards each or 1,000 files of 100 cards each. This consideration discourages techniques that result in a strong efficiency or main storage capacity dependency on the "size and shape" of the file system. Of course, the magnitude of the file system size will affect the operation, but arbitrary restrictions such as "no more than 64 files on a device" would be discouraged unless essential.
5. Removable volumes.
   It has become common to differentiate between the I/O mechanism used to record or read information, called a "device", and the physical

medium on which the information is stored, called a "volume". For most drums and many disk units, the device and volume are inseparable. But, for magnetic tape units and many of the smaller disk units the volume, magnetic tape reel and disk pack respectively, are removable. It is intended that the file system include files that are on unmounted volumes (disconnected from an I/O device) as well as mounted volumes. Therefore, a configuration that consists of ten disk units may have a file system that encompasses hundreds of volumes, only ten of which may be actively in use at a time. Since removing and mounting a volume takes several minutes of manual effort, it will be assumed that the "working set" of volumes (volumes that contain files that are actively in use) remains static for reasonable periods of time and is less than or equal to the number of devices available. The fact that volumes are removable and interchangeable (i.e., may be mounted on different devices at different times) does affect the organization of the file system. For example, a scheme that involved linking files together by means of pointers (chained addressing) could require mounting volumes just to continue the path of the chain even though little or no "logical" information was requested from files on that volume. In the worst case, it might be necessary to mount and unmount all the volumes of the file system to locate a desired file. Such a situation should definitely be avoided if not totally eliminated by the file system.

6. Structured file directories and file sharing.
   In a traditional file system, the mapping between the symbolic file name and the corresponding file was accomplished by means of a single Master File Directory. For modern file systems with thousands of files scattered over hundreds of volumes, it became desirable, if not necessary, to form groupings of files by means of Secondary File Directories.[4] These groupings are often used by the system to associate users with files they own (User File Directories). This capability is also available to the user to arrange his files into further sub-groups (libraries) or into separate project-related groupings. Occasionally it becomes necessary for a file to be included in two or more groupings (e.g., accessible by more than one User File Directory) with potentially different access privileges (protection) associated with each grouping. Many of these features that are relatively easy to implement in a traditional file system are complicated by the introduction of independent processors and removable volumes.
7. Fail-safe operation.
   Reliable operation is a very important requirement of a general purpose file system. There are many known techniques for I/O error and systematic backup and salvage procedures that are applicable to this environment. The important problem associated with the multi-computer network is that potential error conditions exist that are not normally found in traditional single computer file systems. For a single computer system, a processor error (including unexpected processor disconnection, i.e., "turning off") is a rare occurrence. Such a situation is remedied by repairing whatever physical hardware is necessary and then running a special "salvager" program to bring the file system into a well-defined operational state. In the environment of a multi-computer network, processors may be connected or disconnected at any time without any awareness by the other processors. To prevent any inconsistent file system operation by the other processors and eliminate the need for usually time-consuming salvage techniques, it is necessary to keep the file system in a well-defined consistent state at all times.

A file system design

The purpose of the remainder of this paper is to apply the organization presented in the File System Design Model section to solve the problems associated with a multi-computer file system network. Discussion of the Access Methods and Input/Output Control System will be omitted. This is necessitated for brevity and consideration of the facts that the Access Methods are highly application oriented, as discussed in a previous section, and that the Input/Output Control System is usually a basic and common component of all Operating Systems. The principal contribution of this model lies in the structure of the four other levels.

Logical file system

To present the goals and requirements of the Logical File System in a brief and demonstrative manner, an example will be used. The reader should refer to Figure 6 for the following discussion. It is important that the peculiarities of the example, such as the choice of file names (e.g., "FILE6" and "DIR4"), not

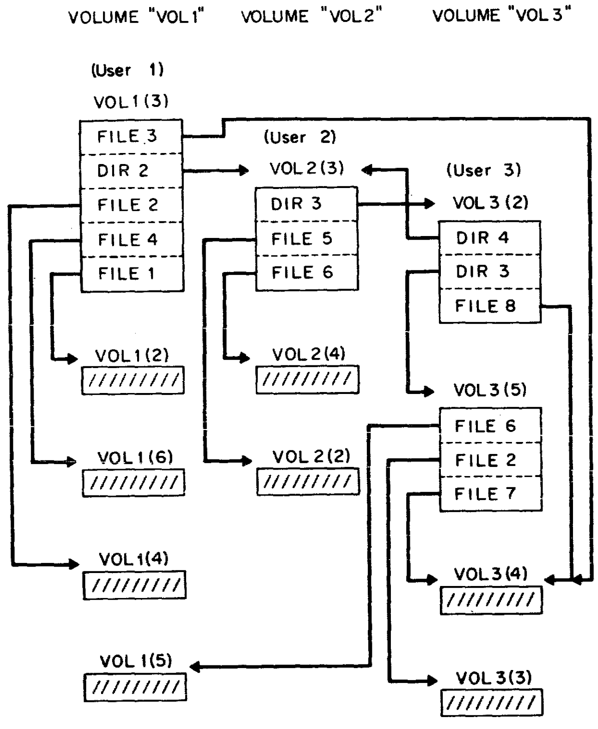


Figure 6—Example of file directory structure (to LFS)

be confused with the general characteristics of the Logical File System.

In Figure 6, there are 12 files illustrated. Associated with each file is an identifier of the form "VOL1(3)". The usage of this identifier will not be discussed until later, in the meanwhile notice that each file's identifier is unique. The 12 files are divided into 2 types, directory files (i.e., VOL1(3), VOL2(3), VOL3(2), and VOL3(5)), and data files (i.e., VOL1(2), VOL1(6), VOL1(4), VOL1(5), VOL2(4), VOL2(2), VOL3(4), and VOL3(3)). The distinction between directory files and data files is *only* a matter of usage, the Access Methods *may* operate upon a directory file in the same manner as a data file, furthermore, all lower levels (e.g., Basic File System) treat *all* files as data files. This factor will be elaborated shortly.

It is the stated function of the Logical File System to map a file name reference into a unique file identifier. This mapping is a function of the requested file name (symbolic file name path) *and* a starting point (base directory) in the file directory structure. In Figure 6, three example base directories are illustrated by associating VOL1(3) with user 1, VOL2(3) with user 2, and VOL3(2) with user 3. Therefore, user 1 references to the file name FILE2 yields the file VOL1(4).

A more complex example can be illustrated by considering the file VOL3(4). User 3 can refer to this file under the name FILE8. Alternatively, it can be referenced by the name DIR3.FILE7. The file DIR3, which is associated with VOL3(5) from user 3's base directory, is interpreted as a lower level directory. Then from file VOL3(5), the file name FILE7 is mapped into VOL3(4) as intended. The file VOL3(4) can be referenced from user 2's base directory as DIR3.FILE8 or DIR3.DIR3.FILE7, for example. From user 1's base directory, it can be referenced as FILE3, DIR2. DIR3.FILE8, DIR2.DIR3.DIR3.FILE7, or even DIR2.DIR3.DIR4.DIR3.DIR3.FILE7.

Two important side effects of the base file directory and file name path facilities are that (1) a specific file may be referenced by many different names, and (2) the same name may be used to reference many different files.

The headings VOLUME "VOL1", VOLUME "VOL-2", and VOLUME "VOL3" are intended to indicate that the 12 files are scattered over 3 separately detachable volumes: VOL1 (containing VOL1(2), VOL1-(3), VOL1(4), VOL1(5), and VOL1(6)), VOL2 (containing VOL2(2), VOL2(3), and VOL2(4)), and VOL3 (containing VOL3(2), VOL3(3), VOL3(4), and VOL3-(5)). If volume VOL2 were detached from the system, user 1 could still reference VOL1(4) as FILE4 and VOL3(4) as FILE3, but could not reference VOL3(4) as DIR2.DIR3.FILE8 nor VOL1(5) as DIR2.DIR3. DIR3.FILE6 since the path would logically require passing through volume VOL2. Furthermore, user 3 is allowed to erase (i.e., remove from file system structure) the file VOL3(4) under the name FILE8, assuming appropriate protection privileges, *whether or not* volume VOL1 is mounted in spite of user 1's reference to file VOL3(4) under the name FILE3.

The Logical File System could be extremely complex if it had to specifically consider the physical addresses of volumes, the device characteristics, and the location of file directories on volumes, in addition to its obvious requirement of searching file directories. These problems are eliminated by introducing the file identifier and the interface with the Basic File System.

The Basic File System processes requests that specify a file in terms of a file identifier consisting of a volume name and index, such as (VOL3, 4), rather than a file name. A sample call from the Logical File System to the Basic File System, in PL/I-like notation, is:

CALL BFS_READ (VOLUME,INDEX,CORE_ADDR,FILE_ADDR,COUNT); where VOLUME is the name of the volume containing the file, INDEX is the corresponding unique index of the file, CORE_ADDR is the main storage address into which data is to be read, FILE_ADDR is the file virtual memory address from which the data is to be read, and COUNT is the number of bytes to be transmitted. Using these features, the *heart* of the Logical File System (ignoring opening and closing files, file access protection, illegal file names, etc.) reduces to the PL/I-like code presented in Table I. It is assumed that the file name has been broken down into an array of path element names (e.g., if name is DIR2.DIR3.FILE8, then PATH(1)='DIR2', PATH(2)='DIR3', PATH(3)=' FILE8', and PATH_LENGTH=3), that BASE_VOLUME and BASE_INDEX initially specify the (VOLUME,INDEX) identifier of the base directory, and that each entry in a file directory is N bytes long and formatted as indicated in the FILE_ENTRY declaration.

Of course, the handling of access (protection) rights, errors, and other responsibilities will make the Logical File System much more complex, but it is important to note that the design and implementation of the Logical File System escapes all physical file organization and device characteristic considerations and complexities.

Basic file system

The Basic File System must convert the file identifier supplied from the Logical File System into a file descriptor than can be processed by the File Organization Strategy Module. A file descriptor is essentially an entry in the Active File Directory (AFD) that contains information such as the volume name, physical location of the file on the volume, and the length of the file. Every file must have an associated file descriptor, but since the number of passive files (i.e., not actively in use) might be very large, the file descriptors are maintained on secondary storage until needed (i.e., file is "opened"). In organizing the secondary storage maintenance of the file descriptors there are several important considerations:

1. There must be a unique file descriptor for each file regardless of how often the file appears in file directories or what symbolic names are used. This is required to maintain consistent interpretation of a file's status.
2. The file descriptor information for a file must reside on the same volume as the file. This is reasonable since if *either* the file *or* its descriptor is not accessible at some time by the system (i.e., unmounted) the file cannot be used, this possibility is minimized by placing them on the same volume.
3. In the same manner that the Logical File System was simplified by using the facilities of the lower hierarchical level, the file descriptors should be maintained in a manner that allows the File Organization Strategy Module to process them as normal files.

These problems are solved by the use of the Volume File Descriptor Directory (VFDD). There is a single VFDD for each volume, it contains the file descriptors for all files residing on the volume. The file descriptors are of fixed length and are located within the VFDD positionally according to the corresponding file identifier's index. In order to exploit the facilities provided by the File Organization Strategy Module, the VFDD can be processed by the lower levels as a normal file. It is assigned a unique file identifier consisting of the volume name and an index of 1, in fact the file descriptor for a VFDD is stored (when not in use) as its own first entry. Figure 7 presents diagrammatically the logical file structure of Figure 6

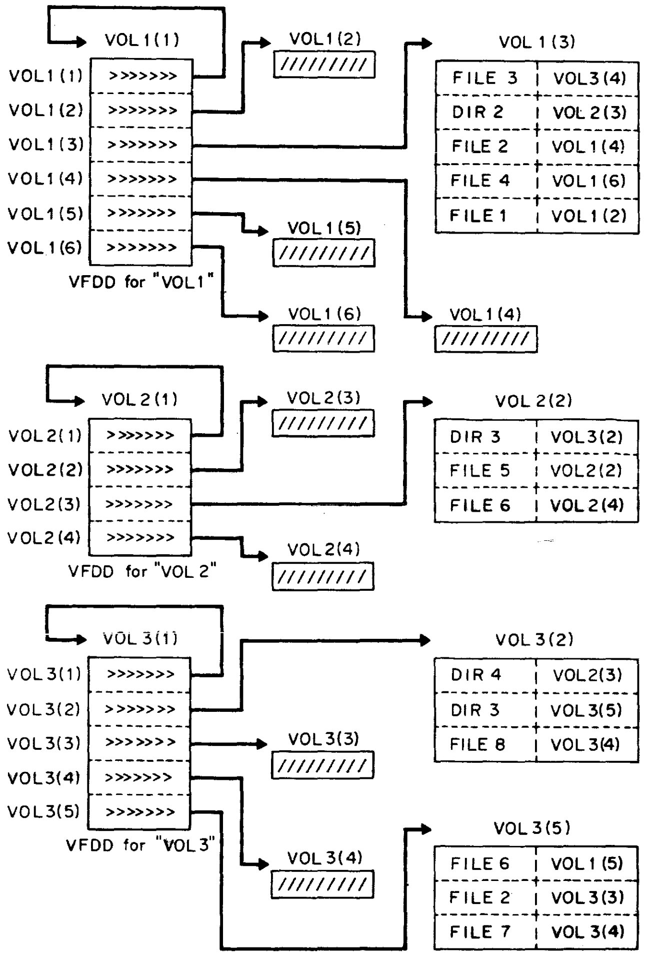


Figure 7—Example of file directory structure (to BFS)

with the added detail of the Volume File Descriptor Directories and File Directory formats.

The File Organization Strategy Module processes requests that specify a file in terms of a file descriptor (the entry extracted from the VFDD) rather than a file name or file identifier. A sample call from the Basic File System to the File Organization Strategy Module, in PL/I-like notation, is:

CALL FOSM_READ (DESCRIPTOR,CORE_ADDR, FILE_ADDR, COUNT); where CORE_ADDR, FILE_ADDR, and COUNT have the same interpretation as discussed above.

The primary function of the Basic File System reduces to the single request:

CALL FOSM_READ (VFDD_DESCRIPTOR, DESCRIPTOR,M*(INDEX-1),M); where VFDD_DESCRIPTOR is the descriptor of the VFDD associated with the volume name supplied by the Logical File System as part of the file identifier, INDEX is from the specified file identifier, M is the standard length of a VFDD entry, and DESCRIPTOR is the desired file descriptor.

The Basic File System performs several other tasks, such as protection validation and maintenance of the core-resident Active File Directory that enables efficient association between a file's identifier and descriptor for files that are in use (i.e., "open"). But, as in the Logical File System, the domain of the Basic File System is sufficiently small and narrow that it remains a conceptually simple level in the hierarchy.

File organization strategy modules

The Logical File System and Basic File System are, to a great extent, application and device independent. The File Organization Strategy Modules are usually the most critical area of the file system in terms of overall performance, for this reason it is expected that more than one strategy may be used in a large system. Only one strategy will be discussed in this section, the reader may refer to the papers listed in the References[2,12,17,20] for other possible alternatives.

The FOSM must map the logical file address onto a physical record address or hidden buffer based upon the supplied file descriptor information. In the simplest case, the mapping could be performed by including a two-part table in the file descriptor. The first part of each entry would indicate a contiguous range of virtual file addresses, the second part of each entry would designate the corresponding physical record address. It has been assumed, however, that all file descriptors have a specific length, whereas the mapping table is a function of the file's length and is potentially quite large. Therefore, it is not feasible to include the entire mapping table as part of the file descriptor. One of the most powerful file organization strategies utilizes index tables, Figure 8 illustrates such an arrangement.

In this example it is assumed that each file is divided into 1000 byte physical records. A file can be in one of several index arrangements depending upon its current length. If the file's length is in the range 1 to 999 bytes, the file descriptor contains the address of the corresponding physical record. If the file is between 1000 and 499,999 bytes long, the file descriptor specifies the address of an index table located on secondary storage. Each entry of the index table (assumed to require 2 bytes) designates the physical address of a block of the file (blocks are ordered by virtual file addresses: 0–999, 1000–1999, 2000–2999, etc.). Furthermore, for files greater than 500,000 bytes, but less than 250,000,000 bytes, there are 2 levels of index tables as illustrated.

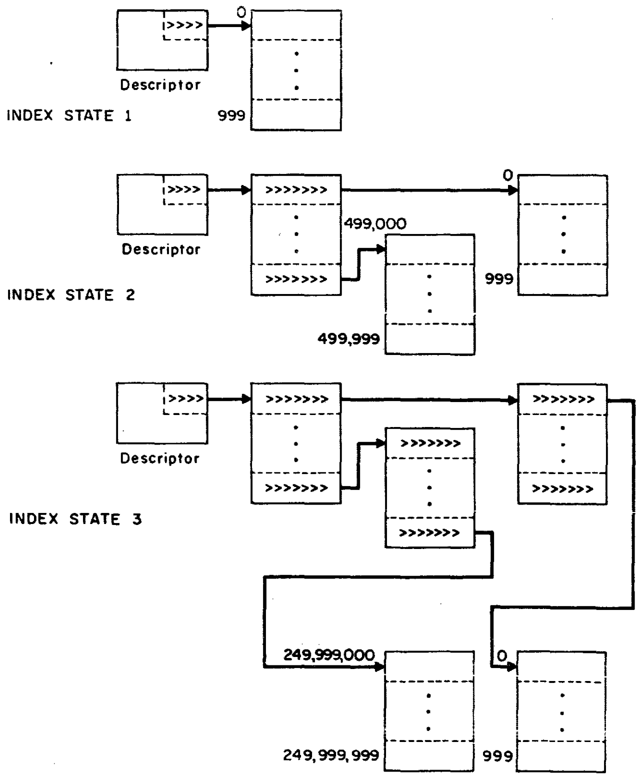


Figure 8—Example of file organization strategy

This strategy has several advantages. Under the worst conditions of random access file processing only from one to three I/O operations need to be performed. By utilizing several hidden buffers for blocks of the file as well as index tables, the number of I/O operations required for file accesses can be drastically reduced.

Device strategy modules

The Device Strategy Modules convert "logical I/O requests" from the File Organization Strategy Modules into actual computer I/O command sequences that are forwarded to the Input/Output Control System for execution. The Device Strategy Modules handle two rather different types of requests: (1) read or write blocks, and (2) allocate or deallocate blocks.

When a request to transfer a large portion of a file (10,000 bytes for example) is issued, it is unlikely that a significant amount of the needed blocks are in hidden buffers. It will, therefore, be necessary to request I/O transfer for several blocks (e.g., about 10 blocks if each block 1000 bytes long). The FOSM will generate logical I/O requests of the form: "read block 227 into location 12930, read block 211 into location 13930, etc." The DSM must consider the physical characteristics of the device such as rotational delay and "seek" position for movable heads. It then decides upon an optimal sequence to read the blocks and generate the necessary physical I/O command sequence including positioning commands. The Input/Output Control System actually issues the physical I/O request, error retry, and other housekeeping as discussed earlier. The detailed strategy for choosing the optimal I/O sequence is, of course, very device dependent and will not be elaborated here.

The function of automatic block allocation is somewhat more complex since it involves several separate factors. Before describing the implementation of the mechanisms, it is wise to review the desired characteristics:

1. A file is allowed to grow in size, the FOSM will request additional blocks for the data portions of a file or its index tables, as needed.
2. Common direct access devices contain from 8000 to 32000 separately allocatable blocks, thus it is not feasible to store all allocation information in main storage.
3. Since two independent processors may be writing new files on the same volume at the same time, it is necessary to provide interlocks such that they do not accidentally allocate the same block to more than one file, yet not require one processor to wait until the other processor finishes.

These problems can be solved by use of a special Volume Allocation Table (VAT) on each volume. In this scheme, a volume must be subdivided into arbitrary contiguous areas. For direct access devices with movable read/write heads, each discrete position (known as a "cylinder") covers an area of about 40 to 160 blocks. A cylinder is a reasonable unit of subdivision. For each cylinder on the volume, there is a corresponding entry in the VAT. Each entry contains a "bit map" that indicates which blocks on that cylinder have not been allocated. For example, if a cylinder consists of 40 blocks, the bit map in the corresponding VAT entry would be 40 bits long. If the first bit is a "0", the first block has not been allocated; if the bit is a "1", the block has already been allocated. Likewise for the second, third, and remaining bits.

When the FOSM first requests allocation of a block on a volume, the DSM selects a cylinder and reads the corresponding VAT entry into main storage. An available block, indicated by a "0" bit, is located and then marked as allocated. As long as the volume remains in use, the VAT entry will be kept in main storage and blocks will be allocated on that cylinder. When all the blocks on that cylinder have been allocated, the updated VAT entry is written out and a new cylinder selected. With this technique the amount of main storage required for allocation information is kept to a minimum (about 40 to 160 bits per volume), at the same time the number of extra I/O operations is minimized (about one per 40 to 160 blocks of allocation).

The problem of interlocking the independent processors still remains. As long as the processors are allocating blocks on different cylinders using separate VAT entries, they may both proceed uninterrupted. This condition can be accomplished by utilizing a hardware feature known as "keyed records" available on several computers including the IBM System/360. Each of the VAT entries is a separate record consisting of a physical key area and a data area. The data area contains the allocation information described above. The key area is divided into two parts: the identification number of the processor currently allocating blocks on that cylinder and an indication if all blocks on that cylinder have been allocated. A VAT entry with a key of all zeroes would identify a cylinder that was not currently in use and had blocks available for allocation.

There are I/O instructions that will automatically search for a record with a specified key, such as zero. Since the device controller will not switch processors in the midst of a continuous stream of I/O operations from a processor (i.e., "chained I/O commands"), it is possible to generate an uninterruptable sequence of I/O commands that will (1) find an available cylinder by searching the VAT for an entry with a key of zero and (2) change the key to indicate the cylinder is in use. This thus solves the multi-processor allocation interlock problem.

CONCLUDING COMMENTS

To a large extent file systems are currently developed and implemented in much the same manner as early "horseless carriages", that is, each totally unique and "hand-made" rather than "mass produced". Compilers, such as FORTRAN, were once developed in this primitive manner; but due to careful analysis of operation (e.g., lexical, syntax, and semantic analysis, etc.), compilers are sufficiently well understood that certain software companies actually offer "do-it-yourself FORTRAN kits". Since modern file systems often outweigh all other operating system components such as compilers, loaders, and supervisors, in terms of programmer effort and number of instructions, it is important that a generally applicable methodology be found for file system development.

This paper presents a modular approach to the design of general purpose file systems. Its scope is broad enough to encompass most present file systems of advanced design and file systems presently planned, yet basic enough to be applicable to more modest file systems.

The file system strategy presented is intended to serve two purposes: (1) to assist in the design of new file systems and (2) to provide a structure by which existing file systems may be analyzed and compared.

ACKNOWLEDGMENTS

The author acknowledges the many long and often heated discussions with his colleague, Mr. Allen Moulton, from which many of the basic ideas for this file system design were molded.

Many colleagues generously contributed their time, energy, and criticism to help produce this final document. Special thanks are due to Prof. John J. Donovan, Prof. David Ness, and Prof. Robert M. Graham, as well as, Stephen Zilles, Ben Ashton, Hoo-min Toong, Michael Mark, Derek Henderson, Norm Kohn, and Claude Hans.

REFERENCES

1 R E BLEIER
Treating hierarchical data structures in the SDC time-shared data management system (TDMS)
P A C M 1967

2 F J CORBATO et al
The compatible time-sharing system
M I T Press Cambridge 1962

3 R C DALEY J B DENNIS
Virtual memory, processes and sharing in multics
C A C M May 1968

4 R C DALEY P G NEUMANN
A general purpose file system ofr secondary storage
Proc F J C C 1965

5 J B DENNIS
Segmentation and the design of multi-programmed computer systems
J A C M October 1965

6 E W DIJKSTRA
The structure of the 'THE' multiprogramming system
A C M symposium on operating systems principles
Gatlinburg Tennsesee October 1967

7 E W DIJKSTRA
Complexity controlled by hierarchical ordering of function and variability
Working paper for the NATO conference on computer software engineering Garmisch Germany October 7-11 1968

8 P J DIXON DR J SABLE
DM-1-a generalized data management system
Proc S J C C 1967

9 IBM CAMBRIDGE SCIENTIFIC CENTER
CP-67/CMS program logic manual
Cambridge Massachusetts April 1968

10 IBM CORPORATION
IBM System/360 time sharing system access methods
Form Y28-2016-1 1968

11 S E MADNICK
Multi-processor software lockout
P A C M August 1968

12 S E MADNICK
Design strategies for file systems: a working model
File/68 international seminar on file organization
Helsingor Denmark November 1968

13 D B NELSON R A PICK K B ANDREWS
GIM-1-a generalized information management language and computer system
Proc S J C C 1967

14 B RANDELL
Towards a methodology of computer system design
Working paper for the NATO Conference on computer software engineering Garmisch Germany October 7-11 1968

15 R L RAPPORT
Implementing multi-process primitives in a multiplexed computer system
S M thesis M I T department of Electrical Engineering
August 1968

16 S ROSEN
Programming systems and languages
McGraw-Hill New York 1967

17 J H SALTZER
CTSS technical notes
MIT project MAC MAC-TR-16 August 1965

18 J H SALTZER
Traffic control in a multiplexed computer system
Sc.D thesis MIT department of electrical engineering
August 1968

19 A L SCHERR
An analysis of time-shared computer systems
MIT project MAC MAC–TR–18 June 1965
20 SCIENTIFIC DATA SYSTEMS
SDS 940 time-sharing system technical manual
Santa Monica California August 1968
21 L H SEAWRIGHT J A KELCH
An introduction to CP–67/CMS
IBM Cambridge Scientific Center report 320–2032
Cambridge Massachusetts September 1968

RTOS—Extending OS/360 for real time spaceflight control

by J. L. JOHNSTONE

International Business Machines Corporation
Houston, Texas

INTRODUCTION

The Real Time Operating System/360 (RTOS/360), a modified version of the standard IBM System/360 Operating System (OS/360)*, was developed by the Federal Systems Division (FSD) of IBM for support of the Real Time Computer Complex (RTCC) during NASA's Apollo spaceflights. RTOS/360 is a real time, multi-tasking, multi-jobbing operating system that extends the basic features of OS/360 and adds additional features to:

- Process real time data
- Provide simplicity of use for the applications programmer
- Ensure fast response system activity (requirements range from one-tenth of a second to one second)
- Improve efficiency
- Provide support for special devices not supported by OS/360
- Provide a fail-safe system
- Increase job shop throughput

The presence of these features in OS/360 does not deter from its basic capabilities; i.e., all the facilities of the current IBM released OS/360 that operate in a standard or non-real time mode of execution are available in RTOS/360.

Some of the major functional areas which were developed at IBM's FSD Houston Operations and added to OS/360 in the formation of RTOS/360 were:

- Independent Task Management
- System Task Capability
- Queue Management
- Data and Time Routing
- Time Management
- Real Time Input/Output Control System
- Data Tables
- Display Formatting Language (DFL)
- Real Time Linkages
- Large Core Storage Support
- Logging
- Simulated Input Control
- Fastime
- Fail-Safe Programs
- Background Utilities
- Houston Automatic Spooling Priority (HASP)
- Statistics Gathering System
- Job Accounting System
- Multi-jobbing

The RTOS environment

Although RTOS/360 can be used in a variety of applications and computer system configurations, it is pertinent prior to discussing its functional areas that we establish the environment in which it was designed to operate, i.e., the Real Time Computer Complex (RTCC), the RTCC hardware, and the RTCC applications programs.

The RTCC

The RTCC is a ground-based computing and data processing complex for NASA's manned spaceflight program. It includes the computer equipment, associated peripheral equipment and program packages to monitor and support—in real time—Apollo missions, simulations, and training exercises.[1]

RTCC is the core of NASA's Mission Control Center

* IBM System/360 Operating System (OS/360) consists of a comprehensive set of language translators and service programs operating under the supervisory control and coordination of an integrated set of control routines. The operating system is designed for use with Models 30, 40, 50, 65, and 75 of Computing System/360.

(MCC) at Houston, Texas. Flight controllers at MCC monitor every phase of a manned spaceflight, from launch through orbit, reentry, and splashdown. During a lunar mission, flight controllers also monitor and support the astronauts during their flight to the moon, the descent to the moon's surface, the liftoff and rendezvous with the mother ship, and the return to earth.

RTCC provides flight controllers with the information they need to monitor the flight and make decisions regarding the mission. This simply means flight controllers sitting at consoles in Houston have precise information in real time such as the status of every on-board system, the condition of the astronauts, their position in space at any desired time up to 40 hours in advance, or the effect that any planned maneuver would have on the spacecraft or the astronauts.

The RTCC is called on to do many things during a mission. Some of the more important or more common requirements include:

- Process radar data during launch and provide flight controllers with present position and velocity
- Provide flight controllers with information on whether or not the spacecraft will achieve orbit
- Process telemetry data and provide flight controllers with vital information such as amount of oxygen remaining in astronaut environmental control system
- Compute the orbital path of the spacecraft from radar data
- Predict the position of the spacecraft at some time in the future
- Compute how and when the spacecraft must accomplish a particular maneuver to change its orbital characteristics
- Compute navigation information to update the Apollo Guidance Computer on board the spacecraft
- Process radar range data and let flight controllers know the spacecraft is on correct lunar transfer flight path, and if not, what maneuvers are necessary to get it back on the correct path
- Monitor the Apollo Guidance Computer during reentry and predict the spacecraft landing point.

In addition to these tasks, and thousands more performed during a typical Apollo mission, the RTCC also has a key role in flight controller and crew training.

To perform the different requirements of the RTCC, each of five IBM System/360 Model 75 computers are assigned a different role and the RTCC is engineered so that these roles can be exchanged at any moment. This unified set of computers allows NASA to run either two actual missions at the same time, two simulated missions at the same time, or a simulated mission and an actual mission at the same time. Figure 1, The RTCC, demonstrates the five systems at work in the latter configuration. In the mission configuration, network data flows into the RTCC from one of the Communications Command and Telemetry Systems (CCATS) at MCC. The data are then sent to the Mission Operational Computer (MOC) in the RTCC, which processes all the real time processing tasks of the Mission, and the Dynamic Standby Computer (DSC), which performs redundancy processing and is ready to function as the MOC, if necessary. In the simulation and training exercise, an Apollo trainer, either at the MCC or Cape Kennedy, is in a closed loop with one of the identical Mission Operational Control Rooms (MOCR). (The other MOCR is being used for the mission in progress.) One simulation computer contains an application program which is generating simulated network data; the other computer is being used as a simulated operational computer. The fifth computer is a standby computer for both exercises; however, it is not idle, but performing job shop checkout for future application program development.

The hardware configurations

There are several System/360 hardware configurations used in the development and execution of the computing systems developed for the real time applications at NASA. Each configuration is supported by a single RTOS/360 system. The configuration used on each of the five computer systems in the RTCC itself consists of a System/360 Model 75 computer with a one-million byte main memory (IBM 2705). (See Figure 2, System/360 Model 75 for Mission Support.) An IBM 2361 Large Core Storage (LCS) acts as a four-million byte extension of main memory as well as a buffering

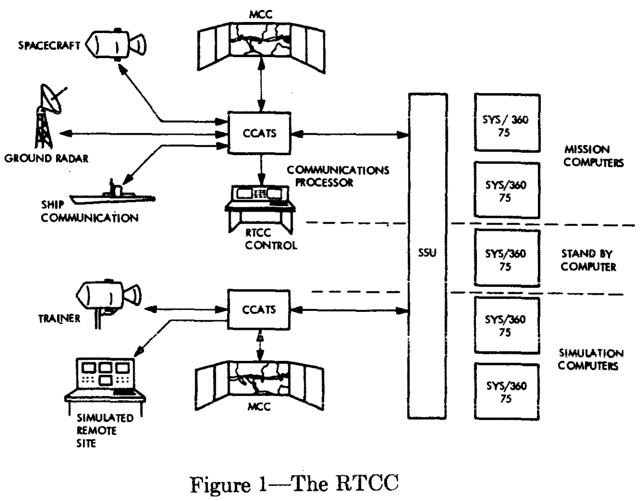


Figure 1—The RTCC

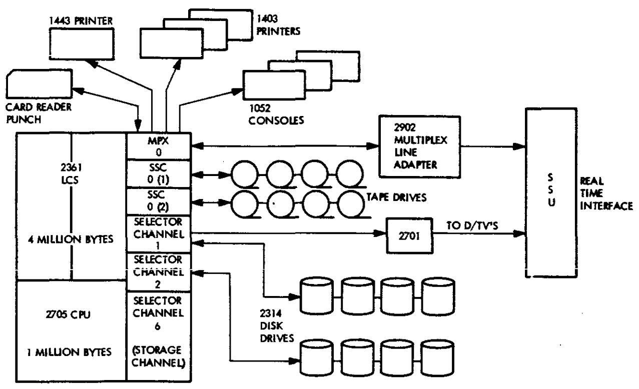


Figure 2—System/360 model 75 for mission support

device for retrieving data and programs from the IBM 2314 disk drives. The IBM 2701 provides a rapid demand response interface to the digital display (D/TV) system in the MOCR and RTCC. Real time acceptance and transmission of large amounts of data and control information are accomplished through the use of the IBM 2902 Multiplex Line Adapter (MLA). A card reader/punch, an IBM 1443 printer, three IBM 1403 printers, two IBM 1052 consoles, and eight tape drives complete the configuration.

Another System/360 Model 75 configuration is used primarily for Simulation exercises. In addition to those devices given in the previous configuration, this Model 75 configuration supports a special Apollo Simulation Processor Channel (ASPC), which receives data from a Multichannel Demultiplexor and Distributor (MDD), an IBM 2260 Display Device, and an IBM 2844 which acts as a control unit for the IBM 2314 disk drives. Several different System/360 Model 50 configurations are also supported by RTOS/360 at the RTCC.

The applications

The applications programming packages used to perform in these various configurations include:

- The Apollo Mission Systems
- The Ground Support Simulation Computer Systems
- The Dynamic Network Data Generation Systems
- The Simulation Checkout and Training Systems
- The Operational Readiness and Confidence Testing Systems.

We've now placed RTOS/360 in its environment; i.e., the RTCC, the RTCC hardware configurations, and the RTCC applications used for real time processing under RTOS/360 control. With this environment in mind, we can now turn to a description of the various functional areas and features designed to extend OS/360 to form RTOS/360. First, let's look at the functional areas.

Functional areas of RTOS/360

Independent task management

In OS/360, all processing is done in conjunction with *units of work* defined as tasks. Tasks are not programs in core storage nor are data for a program a task. A task is a unit of work (programs and data) requiring resources (CPU etc.) to complete its functions. A task exists only when a Task Control Block (TCB) is established and its location is known to the supervisor portion of the operating system. The TCB contains information on such things as pointers to data (I/O), the list of programs needed to operate under the task, the priority of the task, etc. In OS/360, the word "multiprogramming" is replaced by "multi-tasking"; however, the meaning is still the same, i.e., many tasks processing asynchronously—through various paths of logic with the usage of the CPU being switched according to the requirements of the system. (Figure 3, A Task, gives a graphic illustration of a task.)

Some of the characteristics of an OS/360 task are not functionally oriented toward the types of work required to be performed by a real time system. An OS/360 task requires the existence of its creator in order to exist; i.e., it is *dependent* on its creator. This OS/360 concept has been extended within RTOS/360 to include tasks which are *independent* of their creators. This causes a distinction between *dependent* and *independent* tasks. (A dependent task is identical to an OS/360 task.) Therefore, an independent task does not require the existence of its creator in order to exist.

How do the characteristics of an independent task render it more especially suited to real time systems? First, a real time system must be able to receive and process varying data loads rapidly and efficiently. In RTOS/360, an independent task may be defined for each type of data to be processed in real time and be

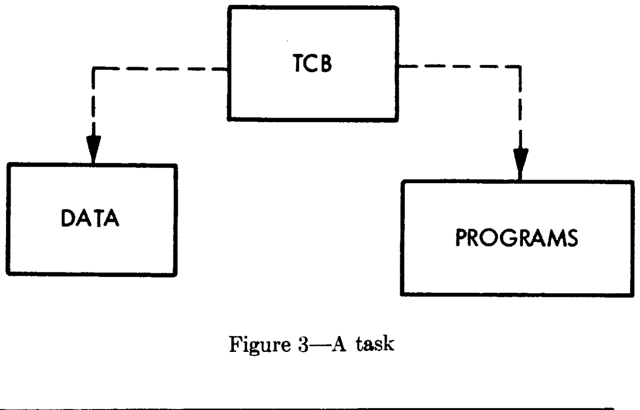


Figure 3—A task

available to receive work at all times even if the data rate is low or random. The major distinction here between dependent and independent tasks is that the independent task will continue to exist in the system when it has no data to process. During this time it is *dormant*. The OS/360 task requires at least one load module executing in order to exist. Each independent task is assigned an area in main core called a resource table. This is a private area that can be used by the programs running under the task. Usually, information is stored in this area which is derived from the processing of earlier data. In this way, the task can "remember" information through periods of dormancy. When data are received by the system for an independent task, they are sent to the task in the form of a request. Each request has its own priority which in turn becomes the task's dispatching priority while processing that request. The OS/360 task has only the priority of its creator. If an independent task is processing a request when another request is generated for it, the new request is enqueued according to its priority. Requests in this queue will be given to the task as it completes the processing of higher priority or older requests.

When an independent task becomes active, it is assigned a unique protect key. This protect key is given to all dependent tasks created by the independent task while processing a request. Therefore, a program running under an independent task or its descendents will be protected from all programs controlled by other active independent tasks or their descendents. Since dependent tasks are assigned the same protect key as their creator's, all tasks of a job step in OS/360 have the same protect key. This is not practical in large real time, multiprogramming systems where many tasks handle various types of data. Independent tasks ensure that unique protect keys will be assigned to unique functions.

Figure 4, OS/360 Task Structure, represents the logical structure of tasks operating as a job step in OS/360. This structure is obviously pyramidal in form. All tasks depend either directly or indirectly on the Job Step Task. The Job Step Task can create dependent tasks (subtasks) which in turn can create tasks dependent upon them, etc. All tasks compete for system resources (CPU, I/O, etc.), and OS/360 awards those resources according to the priority assigned to each task.

Figure 5, RTOS/360 Task Structure, represents the logical structure of tasks operating as a job step in RTOS/360. One can see that a new dimension has been added. The Job Step Task and its subtasks exist as in OS/360 while each independent task forms the basis of another set of tasks which operate independently of and

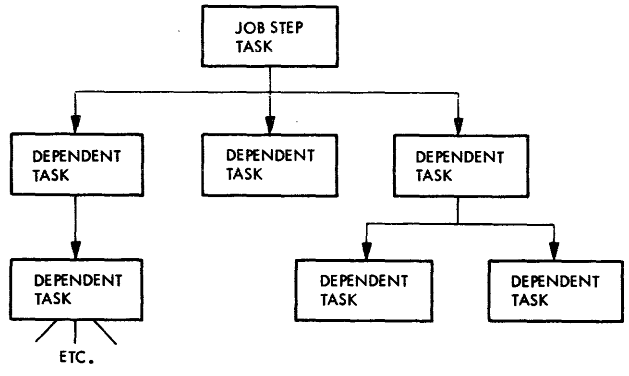


Figure 4—OS/360 task structure

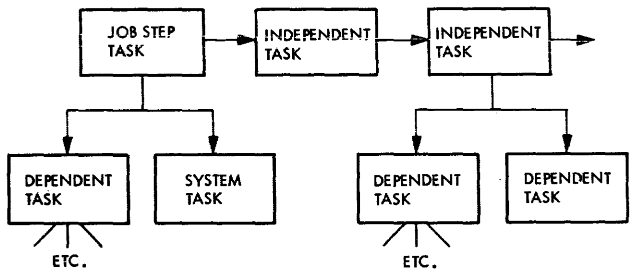


Figure 5—RTOS/360 task structure

parallel to the Job Step Task and each other. This structure is comparable to multi-jobbing in OS/360 with each independent task analogous to the Job Step Task of each active job. However, in RTOS/360, all independent tasks and their subtasks function within a single job step, and all tasks in that job step are awarded the system resources according to their dispatching priority.

System task capability

In the processing of real time data, it was found that many units of work (tasks) were unrelated to an existing task or could be performed asynchronously to existing tasks. These tasks were really tasks of the system. Therefore, a capability was developed in RTOS/360 for these systems tasks. System tasks perform services for the RTOS Supervisor or user-created tasks such as message writing and logging of real time input data. System tasks can be created and returned from within 1/25th the system overhead time required for either an OS/360 defined dependent task or RTOS defined independent task. This reduction in overhead to perform required system services in a real time environment can prove tremendously important during those CPU critical periods of high, real time, data processing.

The large difference in overhead is due to the following:

- All control blocks required for a System Task have been pre-allocated and pre-initialized for efficient utilization.
- The entry point for a System Task is an absolute location instead of a load module name, as is the case for dependent and independent tasks.

Queue management

If an independent task is processing a work request all other requests for that task must be held by the system until the task is ready to begin processing a new request. Therefore, RTOS/360 must build and maintain a queue of work requests which are waiting to be processed by an active independent task. Information concerning each request is held in a Real Time Queue Element (RTQEL). (Figure 6, Independent Task and RTQEL's, shows the logical structure of an independent task and its RTQEL's which are waiting to be processed.) Each active independent task will be processing one work request and that request is represented by the active RTQEL. All other work requests for the independent task are placed in a queue of waiting RTQEL's. This queue is ordered by dispatching priority and, in the case of equal priorities, it is first-in first-out (FIFO). When the task completes processing of the active RTQEL, the top RTQEL in the queue of waiting RTQEL's is made active and is given to the task. If there are no work requests (RTQEL's) waiting for the task, then the task is made dormant and waits in the system for the arrival of new work. All work requests for independent tasks can be optionally placed under queue management controls by directing each RTQEL into a Real Time Queue (RTQ). Each RTQ is created by a user macro instruction which defines the five attributes of the queue:

- Its unique name, which identifies the RTQ.
- Its length, which is the maximum number of RTQEL's to be held in the RTQ before an overflow condition occurs.
- The sequence in which RTQEL's are to be removed from the RTQ and given to independent tasks for processing (dispatching priority, FIFO, LIFO).
- The overflow disposition which identifies the RTQEL to be removed from the RTQ and discarded if the queue overflows (newest, oldest, lowest priority RTQEL).
- Whether the RTQ is currently able to give RTQEL's to independent tasks (enabled or disabled).

Figure 7, Real Time Queue Element Control, gives

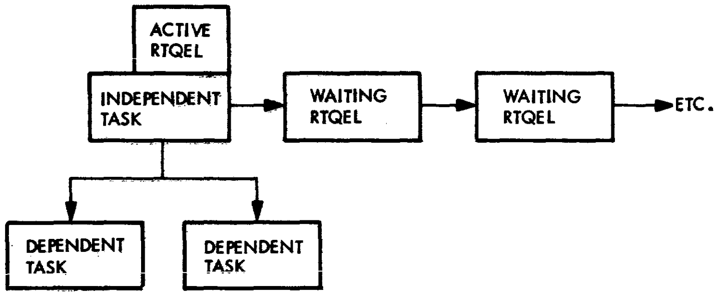


Figure 6—Independent task and RTQEL's

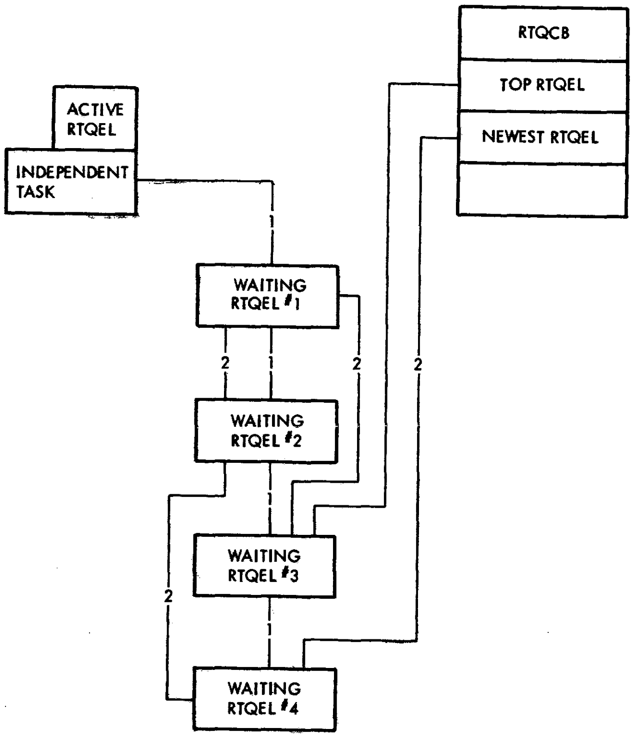


Figure 7—Real time queue element control

an example of the logical structure of RTQEL's controlled by an RTQ. The five attributes and other control information pertaining to the RTQ are held in the Real Time Queue Control Block. In the example, the RTQEL's would be given to the independent task in order one, two, three, four, if they were not controlled by the RTQ. That is the sequence of their relative dispatching priorities. However, the RTQ has a FIFO order attribute; therefore, the RTQEL's will be given to the task in the order three, one, two, four.

If queue management is not used, the RTQEL's for independent tasks in the waiting queues can accumulate indefinitely unless the tasks can process their work requests faster than they are generated. Queue management provides additional controls over the requests in the waiting queues by limiting the maximum number of RTQEL's held for independent tasks. It can also be used to indirectly control the system load by not giving work to an independent task until another independent task has completed processing.

The number and structure of RTQ's is determined entirely by the user. An RTQ can contain work requests for any number of tasks, and any number of RTQ's can contain work requests for the same independent task. The point to be made here is that queue management is very versatile in that it can be used in many ways to regulate the system's work flow.

Data and time routing concept

One of the characteristics of many real time, on-line systems is that they are driven by the arrival of data to be processed and by the passage of time; i.e., some processing is accomplished by the programming system because certain data have arrived while other processing is accomplished because certain reports and displays are required at specific times. Another characteristic found in the RTCC system is that much of the processing is very repetitive; i.e., the same kinds of data come again and again, representing different positions of the spacecraft or different data points for the various telemetered activities that are being monitored. In developing RTOS/360, and the independent task concept, it was recognized that a mechanism was required which would examine all types of input data and cause them to be sent to the appropriate independent tasks for processing. This mechanism is called *data routing*, and it acts as an interface between the hardware interrupt servicing function and the resident nucleus of RTOS/360. Data routing is a simple mechanism which requires only that the applications programs execute a macro instruction to identify the directives to RTOS which link a type of input data to an independent task. When input data is received in the system via the 2902 MLA, RTOS compares the data with the current data definitions established by the applications programs. If a match is found, the data are routed to the independent task that will process them. If no match is found, the message is discarded. Data routing can also be instructed to accumulate a number of data messages (for example, input messages) for the same independent task and generate a request for the task only after the number of messages specified by the user have been received. In this case, all the accumulated messages will be sent to the independent task as one request.

As stated above, work requests may be generated according to the passage of time also. For example, an independent task may be created to control a program which updates the position of a space vehicle every second. The only data necessary to perform this operation is the position of the vehicle at the last second and some orbit and velocity parameters. Since this operation is controlled by an independent task, the necessary data and parameters can be saved in the task's resource table while it is dormant (possibly out of main memory). Since data arrive in a random manner and not necessarily sequentially or on a time cycle, there is no method using input data which will cause requests to be generated for the task which must process a request each second. Therefore, *time routing* must be used to generate the required results. To use time routing, a problem program requests that a certain independent task is to be activated at a certain time or cyclical when a given delta time has elapsed. RTOS/360 routing and time management functions will then activate the independent task at the time requested. If the activation is to be continuous, it is left to the problem programmer to request the activation's end.

The data and time routing functions (which operate under a system task) have been constructed so that their functions can be combined. For example, it is possible to request the accumulation of data under data routing with requests generated by time routing on some specified interval. Each request generated will contain all the data accumulated during the last interval. Another way of using the combined functions is that messages can be accumulated over a timed interval and request generated either when the interval expires or when specified numbers of data messages have been received in the system, whichever event occurs first.

Time management

The System/360 Model 75 computers used to support NASA's real time applications are equipped with a special high-resolution (10μs accuracy) GMT (Greenwich Mean Time) clock and interval timer. In order to provide support for this special hardware, a time management supervisor was developed for RTOS/360 which functions in parallel with the standard OS/360 time management routines. The time management supervisor maintains the system time in a job step pseudo clock, and it controls the setting and interrupt processing from the GMT hardware to keep time and service interval timeout requests from the routing function and other areas of RTOS/360. Additional functions have been added to the time supervisor which provide optional controls over the job step pseudo clock.

Real time input/output control system (RTIOCS)

It was necessary to develop a Real Time Input/Output Control System in RTOS/360 which would service real time input/output requests rapidly and efficiently, perform special device-dependent data manipulation, and support the special real time input/output devices at the RTCC. RTIOCS is comprised of five logical parts discussed in the following paragraphs.

A *real time access method* performs device-dependent data manipulation and sends output messages to the special real time output devices at the RTCC. In addition, standard sequential System/360 output devices (2400 tapes, 1403/1443 printers) may be substituted for the special RTCC devices simply by altering the UNIT designation on cards in the user's input job stream. The real time access method is also used to control the reading of information from the IBM 2250 and 2260 graphic display units. This section of the real time access method functions closely with the graphic display attention control routine, and together, these two areas of the real time I/O control system provide RTOS/360 users the ability to read information from the IBM 2250 or 2260 devices. Writing on the display devices is controlled by the real time access method alone. In this case, the displays are processed as normal real time output requests.

The *real time interrupt servicer and start-stop input routine* provides software control over the real time input devices at the RTCC. The interrupt servicer passes input data to the data routing and logging functions in RTOS/360. The start-stop input routine accepts data whenever an active routing request is present for each particular device. An OS/360 OPEN/CLOSE is not required.

The *digital display control routine* provides centralized and simplified control of the special RTCC devices called digital television displays (D/TV). This control program is entered by user tasks signaling the change of status in one or more of the displays. The current status of the display is updated by the control routine, and it then gives control to the real time access method which updates the actual hardware display.

The digital/TV display control routine provides a software support for the Philco digital/TV display system at the RTCC. This program services all digital/TV display requests, maintains information indicating which displays are currently being viewed and the console which is viewing them, controls the dynamic allocation of the digital/TV channels, and generates work requests for the user tasks which create and update the actual numbers or figures within each display.

Data management—data tables

The large amounts of data required to be accessed by the Mission Systems at the RTCC during spaceflights prompted a careful evaluation of the OS/360 Data Management methods. First, it was found that although the methods were adequate for the environments for which they were designed, the RTCC real time environment produced a unique situation in which system overhead needed to be reduced for reading and writing data. Second, there was no efficient means to enable RTOS independent tasks to share data. Third, due to the critical importance of data in the system, a means to ensure data integrity and consistency had to be developed. Finally, an easy method had to be developed to allow users a simple method of reading and writing data, thereby eliminating the need for complicated coding techniques. The resolution to these RTOS data management problems was the development of control programs to support *data tables*.

Data tables are blocks or arrays of data maintained on direct access devices (2314 disk) in the partitioned format. (Data tables are treated as members of partitioned data sets.) Each data table is identified by its unique EBCDIC name and is defined by its block size and number of blocks. A data table generation program employs these parameters in allocating direct access space for each table, providing the controls required to access it, and storing its initial data in the direct access space provided.

The main utility of data tables is the additional facilities provided by the data table control programs. Here, the standard OS/360 Data Management OPEN/CLOSE logic has been eliminated, thereby increasing the speed at which data can be read or updated. Data can be used commonly by any number of different tasks. The data table programs provide methods of "locking" data tables which ensure data integrity and consistency by delaying any tasks which try to write into a data table until the table is "unlocked." In this way, various portions of a table can be read through different requests and the user is ensured that no update has taken place between requests.

Functional area—summary

Briefly, we placed RTOS/360 in its environment and outlined the major modifications made to OS/360 in its functional areas of Task Management, I/O Management, Time Management, and Data Management to extend it for real time spaceflight control. In addition, we have shown the addition of two new functional areas, Routing and Queue Management, which add additional controls necessary for RTOS/360 to efficiently perform the strenuous requirements of real time processing. However, RTOS/360 development does not end here. Experience had taught us that many additional features and facilities would be necessary in an operating system to process and develop real time programming packages. These features are outlined in the following section.

Special features and facilities of RTOS/360

Large core storage support

The IBM 2361 four-megabyte Large Core Storage (LCS) is supported in three modes of operation by RTOS/360. The first mode is to use the LCS as a means for *improving job shop operations*. This is accomplished by: (1) using the LCS as assembler work space instead of tapes or disks, thereby improving assembler execution time; (2) using the LCS as work storage for compilers to allow larger compilations to be performed in main memory, thereby decreasing compile time and increasing job throughput; (3) placing job control information on the LCS, thereby job throughput is increased; (4) using the LCS as a system residence device for nonresident operating systems programs, thereby giving faster access to them and increasing throughput.

The second mode is to use the LCS as an *addressable extension of main memory*. This is especially applicable to large applications packages being developed on the one-half megabyte main memory System/360 Model 50's.

The third mode of operation was initiated by the fact that it was known from the initial development of the Apollo mission application package that the package would exceed the capacity of main memory and the LCS. (The Lunar Landing Mission exceeds six megabytes.) Therefore, an LCS algorithm was developed that dynamically allows the funneling of data and programs into main memory (see Figure 8, Allocation of Main Memory). Basically, this dynamic LCS allocation means that the LCS is used as a high-speed dynamically changing residence device for load modules and data tables which are heavily used but which cannot be contained in main storage for the duration of the need for them. A load module or data table will be put on the LCS when it is requested and is not presently on the LCS. As long as the load module or data table is frequently used, it will be retained on the LCS; when it appears that the load module or data table is no longer required on the LCS, it may be replaced with another load module or data table.

It is possible to identify load modules and data tables with such low response requirements that they need never be placed on the LCS, i.e., residence on a direct access device is sufficient. Conversely, some load modules and data tables are very critical; therefore, these may be permanently "locked" on the LCS.

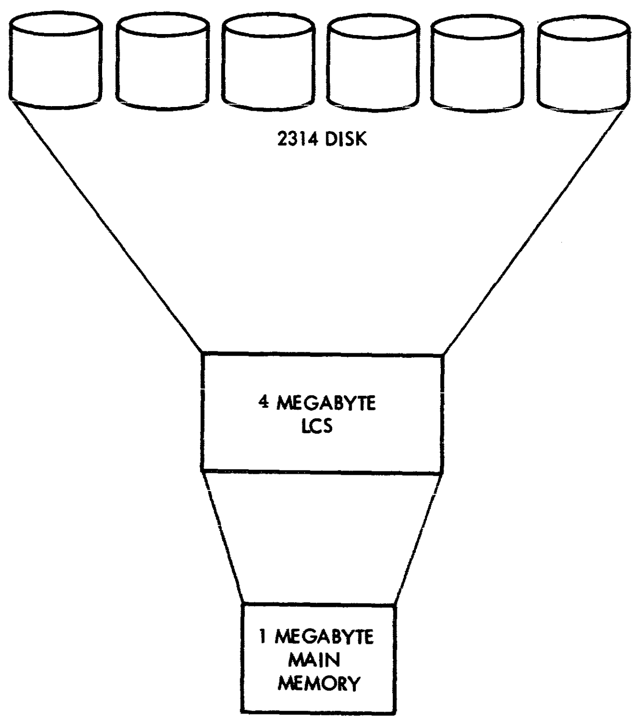


Figure 8—Allocation of main memory

To support the third mode of LCS operation, a Large Core Storage Access Method (LCSAM) was developed to provide the RTOS control program with a facility of moving blocks of storage from the LCS to main storage or from main storage to LCS. LCSAM will perform the data move either with the normal System/360 instruction set or by performing an I/O operation through the storage channel, depending on the size of the block of data.

Real time linkages

Two problems encountered in large real time systems required the development of a feature in RTOS/360 called Real Time Linkages. The first problem pertains to the fact that the system library subroutines referenced by standard OS/360 load modules (programs) must be included within each module when built by the OS/360 linkage editor. This requirement often results in a large duplication of system subroutines present in main core at one time. This duplication can be very wasteful since the amount of main core available is reduced, and the amount of time required to load a module is increased, and the amount of space required to hold the module on a direct access device is increased. Real time linkages solve this problem by allowing load modules to reference common resident reentrant library subroutines.

A second problem pertains to the fact that certain constants (such as the diameter of the earth) used in the real time missions must be identical to all programs and be under close control by the coordinators of the total application mission system. The real time linkage mechanism solves this problem.

By holding task priorities in a common parameter table, the system can be "tuned" by simply changing those priorities found in that single table rather than performing a reassembly of a large number of programs.

Real time linkages resolve all the external references that a load module contains for system subroutines or common parameters when the module is loaded into core for execution. The system subroutines and common parameters are loaded into main core during real time initialization and held there for the duration of the run (job step). Therefore, the addresses of these routines and parameters can be inserted into the appropriate external address constant fields contained in a module as it is loaded so that the cost at execution is no greater than if they appeared in the load modules in the normal fashion.

Logging

In most real time applications, especially those which require post-run analysis, it becomes important to perform some type of recording activity which saves the data received, transmitted, and processed by the system. This feature is referred to as logging in RTOS/360. Logging automatically records all real time input and output messages on magnetic tape. Also, a macro instruction has been provided which will write problem program generated information on the log tape if an application programmer wants a record of selected data or processing results.

Simulated input control

One important factor, which is almost essential in the development of real time systems, is the ability to send simulated input data to the applications programs. In real time environments, it is impossible to employ or always obtain the necessary equipment to produce "live" data for all applications program checkout. To solve this situation, RTOS/360 contains a feature termed Simulated Input Control (SIC), which allows the u er to run his development programs with simulated input data in an attempt to find most interface problems between modules and programming errors prior to final checkout with actual data.

The SIC programs which operate as part of RTOS/360 obtain the simulated input data from cards or tape, or both. All data have a time of receipt associated with each data message which allows SIC to send each one to the data routing function when the time of receipt on the message equals the current internal computer time (job step pseudo clock time). This in turn generates requests for independent tasks which will process the data as if they were a real time message. For convenience, the SIC package has been designed so that magnetic tapes produced by the logging function can be used as SIC input sources without special editing. The SIC programs will pass over all output messages on the log tape and send only the input messages to the data routing function.

Fastime

Another special RTOS/360 function that has become very valuable at RTCC is Fastime. Fastime is often used in conjunction with SIC when testing new areas of the user's system. Its only function is to step the job step pseudo clock when there is no system activity. Fastime operates as the lowest priority task in the system so that it is entered when there is no other activity. If the Fastime program running under this task determines that there is no further work to be performed before the next routing request, the time management function is signaled to step the pseudo clock to the time of the next routing request. One can see that many hours of computer time can be saved because the system will not wait for the actual passage of time to generate a time queue if the system becomes inactive, as time queues will be generated immediately when idle CPU time occurs. This function is further enhanced in SIC runs because the SIC programs use time queues in determining the exact moment a message is to be sent to data routing. Therefore, in a SIC run, time may be stepped to the time of the next data message. This message will be immediately sent to data routing and then to a task for processing. In this way, simulated data messages can be given to tasks as fast as the tasks can process them, thereby reducing the actual computer time to test new programs. By using Fastime with SIC, the checkout of an 80-minute orbit can be performed in about 10 minutes. Fastime and simulated input control have no place and are not used when the system is performing its real time production work. These functions are used only in testing new versions of the application systems.

Display formatting language

There is a large variety of display devices at the RTCC that have different internal format requirements.

There is a high probability of change in these devices, their internal formats, and the displays shown on them. This changeable character of the display devices increased the need for a series of display formatting programs. To meet this need, RTOS/360 programmers designed and developed a versatile display formatting language (DFL) which isolates the applications programmers from the unique characteristics of each display device and the internal format changes resulting from modifications to those devices.

The display formatting process consists of two steps. First, the user must define his display by assembling the DFL *format* macro instruction with his program. The format macro instruction expands into a "format statement" or character stream when assembled.

This character stream provides the display format controls used by the DFL conversion routines in the building of an actual output block for the desired display. During execution, the applications programmer can prepare output data for a display by executing the DFL conversion macro instruction. When the conversion routines complete processing and return control to the calling program, the data are in converted form and ready to be sent to the device specified by a particular control card (DD card) in the job control language for the job step. The user can subsequently output the data to the display device via the real time access method portion of the real time I/O control system. The conversion routines build output blocks for a particular device. The device is specified by identifying to the conversion routines the DD name of the DD card for the data set. From this information, the conversion routines can identify the particular device which is to receive the converted data and activate the appropriate conversion modules. This means that the DFL package provides complete device independence among those devices supported (see Figure 9, Device Independent Display Language).

Through the simple alteration of control cards in the user's job stream, the user can alter his display devices. This feature can be very valuable when the actual display devices are not readily available. The applications programmers can code and debug their display programs using common or standard devices, such as IBM 1403 printers. When the display devices become available, the programs will be ready for actual production work after changing the appropriate control cards. The devices currently supported by the DFL package are printers: IBM 1403, IBM 1443; display devices: IBM 2250, IBM 2260, Raytheon MCVG, Philco RTCC Digital/TV; plotters: RTCC X-Y Plotboards, RTCC Scribers; and Teletypes. The device independence feature and some of the devices supported by DFL are also shown in Figure 9.

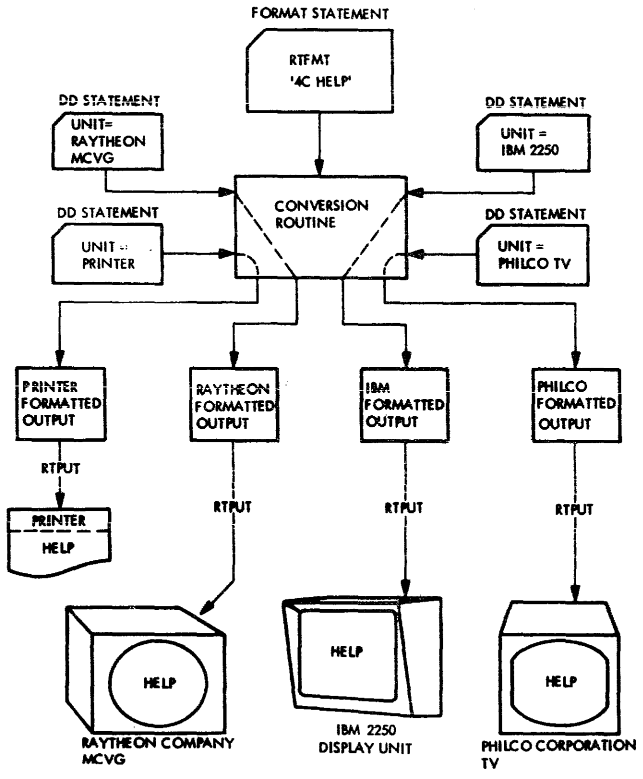


Figure 9—Device independent display language

Fail-safe programs

Because of the critical nature of real time manned spaceflights, it is extremely important that RTOS/360 be able to process abnormal conditions so that it is virtually impossible for a portion of a flight to go unmonitored because of a software, data, or hardware failure. Four areas of software support have been developed and included in RTOS/360 to meet this need: *selectover*, *high-speed restart*, *error recovery*, and *time-out.*

Selectover is performed by exchanging the operational roles of the Mission Operational Computer (MOC) and the Dynamic Standby Computer (DSC) without interruption to the input/output data on the real time interfaces. During Selectover, the integrity of the mission outputs is maintained.

The Apollo mission support system operating under RTOS/360 in a System/360 Model 75 with one megabyte main storage and four megabytes LCS, may be restarted in less than 10 seconds on an alternate Model 75 computer system which may be idle, processing job shop, or performing real time test operations. This is accomplished through IBM Channel-to-Channel Adapters (CCA) which link each combination of two out of five machines. An Initial Program Load (IPL) sequence is generated from a remote console to the proper CCA on the operational computer system, simultaneously enabling the CCA path between the two systems. A special IPL hardware modification enables a restart even if the machine to be restarted is in manual state. All of allocated storage is then transferred over the CCA from the operational system to the selected standby system before resuming in the restarted system. A similar restart can be performed from magnetic tape by creating the tape on an operational computer and carrying it to the standby computer for IPL. (This operation takes about five minutes.)

The error recovery package of RTOS allows the system to recover from errors due to the program errors, hardware malfunctions, or abnormal conditions arising within the system itself. As recovery occurs, appropriate messages and recommendations are printed which indicate the current status of the system. A part of the error recovery activity includes device switching, i.e., if one I/O device fails, RTOS will automatically (or by external signal) select another device of that type. When the control program detects an error condition, an end-of-tape, or when a user requests a device switch, Alternate Device Support (ADS) is invoked to locate an alternate unit of the same device type and perform the necessary adjustments to allow the alternate to replace the primary device. RTOS/360 programs contain built-in logic which allows recovery from a situation where an alternate is unavailable. The computer operator is informed on the console typewriter of all device switching operations. Currently, device switching is provided for the 1052 typewriter, tapes, printers, and 2314 disks.

Certain I/O device failures are such that an interrupt to the CPU is never generated to signal the completion of the I/O operation or an error condition. A software timeout facility exists in RTOS/360 which will check once per second to determine if an I/O operation has not completed in a period of time which is normal for the particular device. When such an occurrence is detected, the I/O operation will be purged and appropriate messages will be printed. Normal use of the device will be attempted on subsequent requests.

Houston automatic spooling priority system

The tremendous development effort required to meet critical mission schedules requires that the computer systems be used to the maximum at all times during job shop operations. It was known from the onset of Apollo development that either a large number of "peripheral" off-line support computers would be required to perform such operations as loading jobs for execution and printing the vast amounts of output (usually large core and LCS dumps) or the Model 75 computers would have to be used to their maximum CPU availability. Since the former was far too expensive, a programming system was developed specifically for RTOS/360 that allowed all the peripheral functions normally associated with off-line support computers to be performed in the single Model 75 CPU. The system was called the Houston Automatic Spooling Priority (HASP) system. HASP acts as a dependent task under RTOS/360 (cohabitates in a single CPU with other RTOS/360 operations) and uses small amounts of primary CPU time to operate the peripheral functions. These functions include transferring the job stream to direct access to await execution, collecting job output on direct access, and printing and punching job output from direct access following job execution. Jobs awaiting any stage of processing (print, punch, or execution) are queued on a priority basis so that the effect of a true priority scheduler is gained not only for normal job execution but for associated peripheral functions as well.

A complete "warm start" capability also exists in HASP so that untimely interruptions of the system will cause no loss of job input or output queued for processing under HASP. Sophisticated operator communications exist that provide control over the number of input job streams, the number of output devices, and the order of job executions.

Background utilities

There are many utility functions in any data processing operation, especially a system which employs disks, that must be performed. These include: dumping direct access volumes to tape, restoring direct access volumes from tape, copying and comparing tapes, labeling tapes, changing volume serial numbers on direct access devices, etc. This is especially true when a large variety of applications systems are under development as in the RTCC. Utility operations usually require the complete dedication of the computer while they are being performed. This dedication was found to be unrealistic from both the cost and time required; therefore, *all* utility operations were designed so that they could operate in "background" under RTOS/360 control as dependent tasks. These background utilities execute asynchronously with the normal job processing and can be initiated and terminated by the computer operator at the console typewriter.

Job accounting system

With the large number of computers being used by a vast array of development groups, it was found that the RTCC required a means to report accounting and system measurement data. This was accomplished by the inclusion of a set of programs called the Job Accounting System (JAS). Since all computer operations at the RTCC are under control of RTOS/360, JAS automatically generates, through punched cards, a data base for three types of reports which are valuable for both the accounting and system measurement purposes. The three report types are:

- A Job Shop Analysis Report which provides job mix and computer system performance statistics.
- A Computer Utilization Report which is used to charge computer time to user.
- A Management Report which provides information on program development costs through statistics on the use of the computer by individual programmers.

Statistics gathering and modeling activities

Each Apollo mission presents the RTCC with a unique set of processing requirements. For example, real time data sources may change in number, arrival rate, or message size. These and other such factors cause changes in the performance of real time computing systems. So that changes do not cause the systems to perform below acceptable limits, performance of current systems is measured and that of future systems is modeled.[2]

To measure the performance of a real time system and monitor its execution, a comprehensive Statistic Gathering System (SGS) was developed. SGS is a program and not a hardware device attached to the computer. It provides an accurate means of measuring performance on RTOS/360 by collecting:

- Timing information on control program services and application programs
- Percentage figures showing how definable system functions use the CPU resource
- Elapsed time figures showing task response time in a multiprogramming environment.

The SGS design for RTOS/360 is patterned after an earlier version used with the Gemini 7094 Executive Control Program.

The Real Time Computer Complex is not a project that is blessed with a firm definition of mission requirements. Results of each mission impose requirements for future missions and, thus, levy new demands for real time support. It is essential to the orderly development of RTCC real time systems to anticipate problems in computer system configuration or system program design that could impair the success of future missions. To analyze future system performance, RTCC uses models written in the language of the General Purpose Simulation System (GPSS/360).

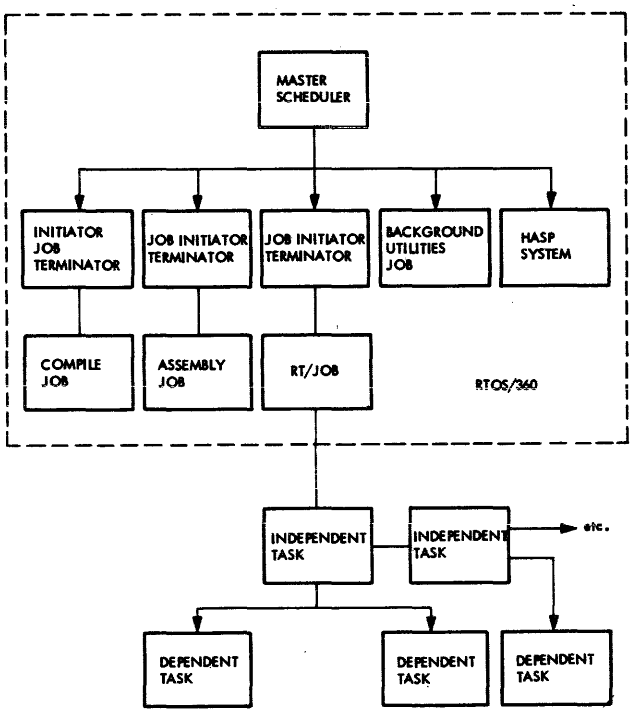


Figure 10—A multi-jobbing/multi-tasking RTOS/360

Information obtained from SGS is used in these modeling activities.

Multi-jobbing in RTOS

Within this paper, it has been shown that RTOS/360 in the real time environment is a multi-jobbing system in the broad sense; i.e., a real time job step can be processing several independent paths of logic (independent tasks which could be termed jobs since they share the system resources) at the same time the multi-tasks are performing, and the background utilities and HASP are also vying for the CPU. However, after careful investigation of statistics obtained from SGS and JAS, it was found that large amounts of time were still spent in the I/O wait state, and that the full computing power of the System/360 Model 75 was not being used. Therefore, the final feature to RTOS/360 was developed—real time multi-jobbing under RTOS/360 control. The RTOS/360 multi-jobbing differs from the OS/360 multi-jobbing in that RTOS/360 does not require partitioned memory nor, of course, a fixed number of tasks. An illustration of RTOS/360 multi-jobbing is shown in Figure 10, A Multi-jobbing/Multi-tasking RTOS/360.

CONCLUDING REMARKS

The development of real time control systems for NASA's spaceflight programs has been an evolutionary process. For the Mercury Program, IBM developed the Mercury Monitor, which performed only real time control and occupied only a small portion of an IBM 7090 computer. Next came the development of the real time Executive Control System for the Gemini program. Executive occupied about 13,000 words of an IBM 7094-II computer. The third system in this evolutionary process was the RTOS/360, which is presented in this paper. RTOS/360, a 150,000 byte system, was the first system not only containing real time control facilities, as in the Mercury Monitor and Executive, but also containing the complete gambit of operating system functions (assemblers, compilers, job shop processing techniques, etc.).

Today, RTOS/360 has not only successfully supported several NASA Apollo Missions, but because of its real time facilities and special features, coupled with the current OS/360 System, is being used by other installations outside the RTCC to meet their special requirements for a real time operating system.

ACKNOWLEDGMENTS

I wish to acknowledge the contributions of all those members of the RTOS departments whose documents and comments aided in the preparation of this paper. Special thanks go to Ray Strecker and Ken Adams, whose technical documents on RTOS were a major source of information. For the encouragement to write the paper, I wish to thank W. D. Pollan.

REFERENCES

1 J JOHNSTONE
A real time executive system for manned spaceflight
Proc F J C C 1967

2 W STANLEY H HERTEL
Statistics gathering and simulation for the apollo real time operating system
IBM Systems Journal Vol 7 No 2 1968

A panel session—On-line business applications

On-line business applications

by JOHN T. GILMORE, JR., *Chairman of Session*

Keydata Corporation
Watertown, Massachusetts

For purposes of background and personal introduction, I would like to begin by stating a few facts about Keydata and its services.

Keydata Corporation was founded in 1959 by Charles Adams and myself and was originally called Adams Associates. Until 1965, when we became the first to offer time-shared business data processing services, our main activity was the design and implementation of real-time, graphic display, and on-line process control systems. That activity is now being carried on by our Keydata Associates Division.

The Keydata system, centered in Watertown, Massachusetts, consists of a private dedicated teletype network that extends as far west as Missouri and south to Delaware. The primary concentration, however, is in New England and Metropolitan New York. Three kinds of computers are used. The UNIVAC 494, utilizing Fastrand and high-speed fixed-head drums, is the time-shared processor and will be duplexed with another 494 before the end of this year. Honeywell DDP-516 computers, used as teletype line monitors and message concentrators, are strategically located geographically within the network to minimize communication costs. An IBM 360 Model 40 is the batch-oriented off-line processor.

Based on the kinds of service we are now providing, the system has a capacity of 800 to 1,000 lines with a response time of less than two seconds. Our present load is about 175 lines with access to more than 630,000 records or 92 million characters. Our basic services are distribution accounting and accounts payable. Certain services are used by a small number of subscribers and other services are under development.

We call our system a "business computer utility" because it provides the businessman with computer power and programs that serve as an efficient tool in operating his company. Like the telephone and electrical utilities, we provide our services through on-line terminals located at his place of business and operated by his employees who are trained in his company's activities and who are not—and need not be—specialists in the service provided by the utility.

Right now, the average businessman would be satisfied to have his conventional data processing needs fulfilled without costing him an arm and a leg and confusing the hell out of his employees. However, once this is accomplished and he realizes what else is possible, he'll roar like a lion. Will we in the computer field be ready for him?

The businessman has as tough a problem to solve as the scientist—perhaps even tougher if one counts his variables and unstable conditions. His basic problem is his data base. He cannot watch it or experiment with it as easily as his brother scientist or engineer can. More often than not, his access to "current" information in his data base is measured in several days to weeks—by which time it is far from being current. However, modern computer technology and on-line communication techniques will enable him to keep his data base current and available in milliseconds. With this kind of luxury he could rapidly become as sophisticated a computer user as his scientific friend. When he does, and when there are many like him, the impact on the business community will probably cause the operations research textbooks to be rewritten and the economists to take a second look at their crystal ball!

What it boils down to is this. On-line business applications, whether on-line to one's own computer or to a computer utility, will provide a dynamically updated data base. Once that occurs, the businessman will be in a position to request the initiation of various operations either at will or automatically. This will provide him and his employees with the timely information essential to the efficient operation of his company. Changes in a data base will automatically cause reactions to other parts of the same data base and, through the use of communications, changes to other data bases, etc.

For example, in performing its invoicing and inventory control function, our system signals the terminal operator when a re-order point is reached based on the quantity just processed in the invoice being prepared. The re-ordering of the item is now being done in the conventional manual way. But the time will soon come when the computer will communicate directly with vendors of the item, compare prices and delivery dates, and order a specific forecast-calculated quantity of the item from the vendor selected. The same re-order example might instead trigger a production order message to another terminal in the plant or perhaps directly to a process control computer. The examples could go on and on, but the main theme assumes a data base that is being dynamically processed.

Among the questions I feel the panel should discuss are:

- What are some of the problems in dynamic data base systems? What protection must be provided to safeguard information? What kind of reliability and availability criteria should there be? What kind of back-up procedures should be employed?
- To the businessman, each successive stage of developing what for him will be the optimum system has to be economically justified. What are some of the major impasses? What are the intermediate payoffs? What kind of time period are we talking about for a sophisticated computer-oriented business community?
- What industries or job groups may suffer from a drastic increase in on-line processing? What government intervention, if any, can be anticipated or perhaps urged?
- Most of us in the computer field see the computer and its power as a valuable tool to the business community. Are we missing anything? For example, will it cause a major cleavage between the unskilled or semiskilled and the bright workers and managers who will use the computer tool to run the plants and offices? Will the leisure class ironically consist of the rich and the unemployed even more so than it does today?

On-line business applications

by CHARLES T. CASALE
Paxon Corporation
Sunnyvale, California

If we look at business applications which are truly on-line today, we see far fewer than were predicted four or five years ago when time-sharing and on-line systems were promised as ultimate solutions. Why is this so? I believe there are several reason, intertwined, which have in turn caused secondary effects. I would like to review some of these briefly.

On-line as a philosophy

To some, on-line reaches the proportions of a religious belief: it is good for everyone, there are ceremonies, there are the articles of faith, and there are the missionaries and prophets. In fact, there are large numbers of people who simply reject onlinism or say that they simply don't need it. At least not now.

Much of the reason for apparent slippage in implementing on-line systems comes from the earlier prophetic statements of dire need. If dire need is assumed, then it can be reasoned that an economic solution is not far behind.

The degree of need

There is real need for on-line, "instant verification" of credit cards at gasoline filling stations to capture voided or stolen cards. So far, we have no systems filling this need since we don't need the systems badly enough to pay the current price. The technology is there. On the other hand, the benefits of having quick retrieval of airline seat availability are worth the cost to the airline. So we have large airline seat reservation systems.

How loud are the demands for on-line inquiry of the weekly payroll file? Or accounts payable? Or for change in status of major sales prospects? The promise of "instant data" in management information systems is proving illusory in many cases; the information just doesn't change that frequently, or to that degree, to warrant the expense and complexity of an on-line system.

The transition period

The transition from an "old style" manual system to a fully on-line automated system is an undertaking as enjoyable as crossing the Italian Alps by elephant in the winter. It can be done, but the participants and bystanders are loath to ever repeat the experience, at least in the foreseeable future. Hardly an issue of the popularized computer magazine goes by without at least one glamour or horror story of the transition period. These periods take time, rub nerves raw, chew up valuable resources that others think could be used elsewhere, and are full of surprises. When finished, the accomplishment is indeed impressive . . . so much so that we are reluctant to make any changes to the new system. This brings us to some other matters:

Who is going to do the system?

First, there is the jurisdictional problem. Secondly, there is the resources problem. Where do we find skilled people to plan and implement the system, who will maintain it, how will they explain it to its users? The shortage of skilled practitioners need not be restated. The shortage is responsible for delays in putting business systems on-line. Once installed, how do we keep it current? With what tools? Can the system grow gracefully, or must it be abandoned when it gets modestly larger?

Profits and returns

After the fact, what are the real bottom-line profits that the system gives me? Can I show a return on this sizeable investment comparable to my other business investments? Are the marginal benefits being used to justify more than their proportionate costs?

Where do we go from here? What are the trends?

It would be useful to look at the major influencing factors that will accelerate or impede progress towards more and better on-line business systems. Some of these are in conflict, others are synergetic.

Data transportation costs

For a given quantity of data, rates decrease slowly. For the same costs, the quantity of data that can be pumped over common carriers increases much faster. Consequently, the threshold of economics for cost justification remains relatively constant; but once justified, the marginal costs of transmitting additional data are small. This will be changed when the full impact of the Carterfone decision is implemented. The result will be a lower threshold caused by reduced termination costs.

Central storage costs and central computer hardware costs

These continue to decrease relatively, and each year sees a lower threshold of economic entry. However, logic and electronic costs are decreasing at a faster rate than storage costs. Consequently, main frame manufacturers are adding more logic to a given storage size to do a more sophisticated job. We are beginning to see parts of software systems being hardwired into computers as a cheaper way to get the job done. This same phenomenon of rapidly declining logic costs has given rebirth to the mini-computer.

The mini-computer

What it lacks in capacity it makes up in muscle. With storage being the majority hardware expense and more logic being condensed onto one semi-conductor chip, the present types of mini-computers will tend to become off-the-shelf components marketed by distributors and produced by the memory houses. What this means to on-line business systems is smaller subsystems operated relatively independent of the parent data base. We are already seeing specialized subsystems based on "standard" mini-computers.

Specialized subsystems

Data acquisition, data distribution and highly structured repetitive tasks can be performed more cheaply with specialized subsystems than by a general-purpose system. This is true only because of the continued decreasing cost of logic and storage. We have seen the success of this in the keypunch replacement area, where no loss of system capability is experienced in using key-to-tape devices instead of the more generalized and flexible keypunches.

The programmer gap

Much has been said, and much is being done about it. Meanwhile, there just aren't enough of them around to get done all of the jobs that are on the drawing boards. And it seems that the gap doesn't close as fast as many expect.

Programming costs

These are increasing because of personnel shortages and because there is a Parkinson effect about any productivity increase made in programming. While today's programmer can produce (via higher-level languages) ten times what he could ten years ago, the job is enlarged to an even greater multiple. Documentation demands are considerably greater and productivity increases have been significant.

Consequences of these trends for on-line business systems.

Large data bases will continue to prosper and flourish, as will their extensions, the time-shared terminals. In addition to this, an entire sub-industry will mushroom based on successful exploitation of the disparity in trends among decreasing hardware costs, rising software costs, and relatively level transportation costs. The results will be specialized subsystems, ranging from a few thousand dollars to several hundred thousand dollars. They will be application-oriented, limited-purpose and highly cost-justifiable.

On-line business applications

by MARTIN GREENBERGER

The Johns Hopkins University
Baltimore, Maryland

The federal government is destined to play a key role in the future development of on-line business systems and the structure of the new industry growing up around them. Six points of contact already are evident:

1. Anti-trust action (real and implied) as in the current cases against IBM and AT&T.
2. Inquiries on the practices and policies of the communications common carriers as in the recent FCC inquiry on the relationship between computers and communications, and the now discharged Task Force on Telecommunications appointed by Lyndon Baines Johnson.
3. Hearings on privacy and associated rights of the individual as conducted last year by Congressman Gallagher and Senator Long.
4. Legislation on copyrights and other possible mechanisms for protecting computer software.
5. Encouragement of standardization and economy measures as in the Brooks Bill and intended activities by the National Bureau of Standards, the General Services Administration, and the Bureau of the Budget.
6. Direct and indirect subsidies of development through research grants, purchase of services and equipment, and support of education.

A great deal of government participation and involvement has gone on in the past year, and every indication is that its pace and scope will intensify in the years ahead. It is helpful to review the main issues and examine the actions taken to date as a basis for distinguishing trends and anticipating future policy and legislation. The intelligent businessman today cannot afford to become fully preoccupied with his own plans and problems in a framework shaped solely by competitive forces. The role and influence of the federal government should be studied and understood.

On-line business applications

by WILLIAM M. ZANI

Harvard Graduate School of Business Administration
Allston, Massachusetts

On-line computing systems are a relatively recent development in commercial computer technology. Most of the experience to date with the use of such systems has been gained at universities, large governmental organizations and a few pioneering businesses like American Airlines, Westinghouse and Keydata Corporation. Today, from the literature and recent announcements of computer plans, it appears that on-line systems are ready to be implemented in many organizations to perform a large variety of tasks. There are experts predicting that by 1970 the majority of computer systems sold will be performing some on-line functions. Huge growth is expected to occur in computers capable of operating on an on-line or time-shared basis.

Much of the growth in usage of on-line systems will be economically justified. The on-line systems have the potential to dramatically change business data processing and the manner in which business will be conducted. On-line business systems can be used to improve decision making, the operations of a business and customer service. Certain aspects of decision making can be improved with on-line systems because a problem solver is able to test on-line to the computer several alternatives and get immediate feedback of results. The interaction of a problem solver with a computer may provide an understanding of a problem and its solution that is not possible to achieve with the long delays in the computer-generated answer cycle of systems without on-line capabilities.

Operations of a business improve with on-line computer systems because the reduction in data processing delays can be used to reduce inventories, provide more efficient distribution systems and more effective production schedules. Westinghouse has implemented an on-line inventory control system for its distribution network, and was able to close seven warehouses and dramatically reduce the level of inventory required. On-line systems have the potential of improving customer service by insuring a complete stock of goods and enabling faster response to customer requests. American Airlines was the first to use an on-line passenger seat reservation system. The company expected increased passenger sales to result from its ability to process customer requests more speedily and accurately.

While much of the growth of on-line computer use will be justified, a significant portion of on-line use will not and cannot possibly be economically justified. Many companies will repeat the same mistakes in switching to on-line computers that were made when computers were first introduced into their organizations. Many companies could not economically justify their original computers, but they purchased them because:

- their competition had a computer and they felt they could not compete successfully without one;
- the companies wanted to keep up-to-date with the latest management techniques, and
- the computer offered many subjective advantages, such as quicker and more information.

Companies, because of the potential advantages of on-line systems, are in danger of being caught up in a whirlwind of unnecessary and uneconomical change. While on-line business systems can be extremely valuable, this value is not automatically achieved nor is it applicable to every business situation. Before any company decides to implement an on-line system, it must examine the economics of the specific applications; otherwise, expensive mistakes can be made. The reasons for this follow.

On-line computing systems do not automatically improve operating performance. They generally increase the speed of information flow in an organization, but if this speed is not used or needed it has no economic value. On-line systems will generate operating savings only if the reduction in information delays are meaningfully integrated into a management process, and if the reduced information delays can improve a company's operations.

On-line systems do not necessarily provide a marketing advantage vis-a-vis competition which does no use such systems. The nature of the industry, tht competition and business may not provide an oppore tunity to gain a competitive advantage. It would be erroneous to impute the competitive advantage gained in the airline industry by American Airlines to companies that use on-line systems in all industries. The generalizations of competitive advantage from on-line systems must be made carefully and only after considering whether: (1) on-line systems can improve cus-tomer service, and (2) the improved service will result in increased sales. It is therefore not necessary to use on-line systems simply because competition is doing so. Depending on the nature of the industry, a company using traditional data processing equipment can satisfactorily compete with a company using on-line systems.

In an evaluation of whether or not to implement an on-line system, it is not sufficient to state that on-line systems are better simply because they provide more rapid information flow. It must be shown how and why the quicker information can be used, and the potential savings that can be related to the faster availability of information. If a company does not have a clear idea of the specific benefits an on-line system will provide and how these benefits are to be achieved, on-line systems should be evaluated on a cost displacement basis; that is, the method of data processing that performs the required functions at the lowest cost should be selected.

A panel session—Computers and the underprivileged

JAMES H. BURROWS, *Chairman of Session*
The Mitre Corporation
Bedford, Massachusetts

Computers and the underprivileged

by MILTON BAUMAN
Price Waterhouse and Company
Philadelphia, Pennsylvania

A group from the Delaware Valley Chapter of ACM have joined together to form the Urban Education Committee. Members of the committee believe that, in these days of urban crisis, the data processing industry offers a unique opportunity to the disadvantaged to become involved in the mainstream of the American way of life.

After several abortive attempts to determine where to employ its professionalism, the Committee has decided upon several projects recommended to it by different groups within the city of Philadelphia. These projects include providing counseling on data processing to the Pennsylvania Employment Service and Philadelphia high school counselors; providing advice and counsel to the Board of Education on the data processing curriculum taught in high schools; providing data processing consulting to the Philadelphia Urban Coalition; and, most important, running a computer operator training course in a Philadelphia high school.

The computer operator training course is being given to 20 high school seniors from Thomas Edison High School. The Philadelphia Board of Education was asked to provide the students. We requested that the students not be college bound, but be of such caliber that would probably insure the success of the training program. The Board of Education, in turn, asked the Principal of Thomas Edison to select the students whom he believed showed leadership qualities and the intelligence necessary to successfully complete the program. The 20 students finally decided upon were selected from a group of 48 recommended by the Thomas Edison faculty.

The training program meets Wednesdays and Saturdays. The curriculum is aimed at IBM 360/30 operator training and extensive hands-on training will be provided. The curriculum is divided into two parts. The first sessions deals primarily with training on punched card equipment to provide the students with a background in data processing and operations done on computers. The 360/30 training will concern itself primarily with IBM's Disc Operating System.

The curriculum for the course was developed by a sub-committee composed of three members. We attempted to obtain curricula from other ACM chapters who had gone down this road; but the other chapters had not, when we were beginning the program, formalized their curriculum to the point where we could use it. It became necessary, therefore, for the sub-committee to collect operator training manuals from various computer manufacturing organizations and from private sources.

Developing the curriculum was only one of several problems that were encountered when the program was started. Other problems were finding instructors (paid instructors, not volunteers), obtaining funds for paying for instructors and field trips, etc., finding machine time (both punched card and computer), and, finally, placing the students in jobs after graduation (we were advised that if students were trained and were not placed in jobs, the training would be at best worthless, or even worse than no training at all). I can report, at this juncture, money has been donated, that instructors have been hired, machine time has been donated, and 12 of the 20 students have been placed in various companies throughout Philadelphia and the Delaware Valley.

In summary, the technique used by the Delaware Valley Chapter in becoming involved in urban affairs was by becoming involved—headfirst. We collectively made up our minds to do "our thing" and, with a little urging from the Board of Education, we did it. We had a few anxious moments in getting started without having all of our problems resolved, but the fact that we had to resolve them in a short period of time made us devote that extra amount of effort necessary. If we had held off our program until all problems had solutions, our course probably would not have started in 1968, but rather in 1969 or perhaps never.

Our plans for the future include a similar program in a Camden High School (we do have a number of interested members from RCA at Cherry Hill, New Jersey) and expansion of our Philadelphia program to another high school.

A program for the underprivileged and overprivileged in the Boston Community

by JOHN J. DONOVAN

Massachusetts Institute of Technology
Cambridge, Massachusetts

In response to the needs of the Boston Community, a new direction has been taken in the Lowell School, a school under the auspices of the Massachusetts Institute of Technology, which will offer education to the community for no more than the cost of two bushels of wheat. The program focuses mainly on the needs of two sectors of our community.

One sector is those individuals who are undereducated and underemployed, many of whom are the "underclass" of the ghetto, suffering from the crippling syndrome of education-motivation-employment deprivation. The other sector is those senior men of industry who are very well educated but need retraining in some aspect of recent technology, e.g., lasers, computer systems.

A basic program for the first sector in computer programming was established. The tuition to the school was set at five dollars, a figure low enough to exclude an absolute minimum and yet still be a commitment on the part of the student. The response to our notice in local newspapers, radio and through public relations channels at M.I.T. was overwhelming. For lack of staff to screen applicants, the first five hundred were accepted. Although our largest interest was in providing access to education for ghetto dwellers of limited resources, our inability to screen applications resulted in a net of about 140 hardcore "deprived" out of the total of five hundred. Of these 140, about 70 percent were black.

Five graduate students at M.I.T. provided the instruction and were assisted by ten M.I.T. undergraduate teaching assistants. The plan is to initiate a chain of lectures by asking successful teaching assistants each year to lecture the following year. We focused on the 140 hardcore "deprived" assigning seven of the M.I.T. teaching assistants, most of whom were black, to those students. These students were divided into smaller tutorial sections of ten to fifteen students headed by one of the teaching assistants. Teaching assistants were also available for consultation in the keypunch rooms.

An M.I.T. student-run computer company has offered to assist in a placement service for these students. We feel that follow up to the program is very important and is presently very weak.

An advanced systems programming course was offered to the second sector of the community, the highly educated sector. We accepted 80 into that program. We feel that these programs will tend to complement each other in that the advanced program will be taught to people who may later assist or influence the hiring of those in the other program.

The basic program will be expanded to include courses in computer maintenance, Boolean algebra, basic business algebra and other practical courses. As technology changes business trends change; the program will be modified to fit the needs for training in the changed environment. It is our overall purpose to offer courses broad enough to establish a basis for training in a particular area. We do not pretend to offer a substitute for the broad knowledge acquired from a college education. We do try to offer a program in a specialized area which has two undeniable attractions: job opportunities and subject excitement. Computer programming is such an area.

What the JOBS program is all about

by WILLIAM B. LEWIS

U.S. Department of Labor
Boston, Massachusetts

One of the thorniest problems in America today is that of the habitually unemployed people living within the inner core of our 50 largest cities. For a long time employers and organized labor have written them off as unemployables. The U.S. Department of Labor has, over the years, tried various approaches to these hard-core jobless, with uncertain success.

In January 1968, President Johnson announced a program of Job Opportunities in the Business Sector (JOBS). The new program looked to industry to apply its full resources and "know-how" in cooperation with the Government, to help break the cycle of unemployment of the hard-core by making them permanent productive members of the labor force.

In announcing the JOBS program, the President said he was calling on American industry to establish a National Alliance of Businessmen (NAB) to launch it, help achieve its goals, and advise the Government. Under the leadership of Henry Ford II, NAB was created early in 1968 with leading local businessmen volunteering to spearhead the effort in the 50 largest cities of the country.

The NAB goal for JOBS is to put 500,000 disadvantaged persons in jobs by June 1971, with an interim goal of 100,000 to be placed by June 1969.

The JOBS program involves a commitment by employers to hire workers first and train them afterward—building on the accumulated evidence that initial placement in jobs at regular wages does much more to motivate a disadvantaged individual than a training period before employment with only a promise of a future job. The program puts at the disposal of industry the services and financial support of Government, which experience has shown are essential if the disadvantaged unemployed are to receive the range and depth of services required to help them become productive workers.

The cooperating companies provide jobs and training for hard-core unemployed workers and bear as much of the cost as would be involved in their normal recruitment and training operations. The extra cost of added training, counseling, remedial education, prevocational training, health services, and other specialized support needed to bring disadvantaged individuals to a satisfactory level of productivity and keep them on the job may be offset by funds provided through a Department fo Labor contract. In order to encourage smaller companies to participate, an optional standardized program approach has been developed. Intensive efforts have also been made to give cooperating employers all possible technical assistance in developing plans and formal proposals.

The first-year NAB goal of 100,000 hard-core persons on the job has been reached by the JOBS program ahead of schedule.

A full assessment of the JOBS program results is not possible at this early stage, but it is apparent that the start made is highly promising. The attitude of participating companies is generally either optimistic or enthusiatic, and they concur regarding the validity of the JOBS idea and intent.

The immediate effect of the JOBS program has been to employ those formerly thought to be unemployable. However, the benefits of JOBS are more far reaching. The skills gained through the JOBS program open the doors to advancement to those formerly without hope. Moreover, what the private employer's experience in the JOBS program has taught him about the problems of the hard-core and the possible solutions to their special problems will, in a large number of cases, have a spillover effect on the company's regular training and employment practices.

Computers and the underprivileged

by ALLEN L. MORTON, JR.

Computer Personnel Development Association, Inc.
New York, New York

Statement of objectives

The Computer Personnel Development Association, Inc. (CPDA) is an organization that has been set up to secure openings in the computer field for individuals from ghetto areas. To prepare these people for work in a business environment, CPDA will provide orientation and training courses in data processing. The program is organized by professionals within the computer industry in collaboration with local community development groups who will help select participants for the program, and with industrial leaders who will locate and provide job opportunities for the participants.

The long term objective of CPDA is to establish career paths in the computer industry for our students.

This will be accomplished by providing continuous job training and career guidance in all areas of data processing.

The following points define the broad areas of CPDA's capabilities.

1. *Computer Operations Training Program*—Training ghetto personnel judged capable of completing a training program in computer operations and functioning in this capacity within the data processing area.
2. *Computer Programmer Training Program*—Training similar personnel who are in a position to complete a training course in computer programming and to function as programmers.
3. *Job Placement and Development*—Moving graduates of the above programs from the training phase into jobs which will be identified prior to and concurrent with training.
4. *Career Guidance*—Providing follow up procedures to smooth the students' transition from the training to the business environment.
5. *Related Personnel Services*—Making available to management on a consultant basis more precise selection and training procedures for minority group personnel.

Implementation of the above program will provide an opportunity for untried minority group persons who show a potential for achievement. This program will serve as a source for desperately needed technicians in the data processing field as well as provide a program which realistically meets the job-related directive of the President's Bipartisan National Commission on Civil Disorders.

Our first project is a pilot program to train and place computer operators. This program is limited in scope but can succeed only with the active support of industry.

Experimental and demonstration manpower projects

by JOSEPH SEILER

U.S.Department of Labor
Washington, D.C.

The U. S. Department of Labor's experimental and demonstration (E and D) program seeks to develop and test through actual project operation, new ideas and techniques to meet manpower problems more effectively. Projects focus on the particular problems which impede employment of the unemployed and underemployed and which are not being met effectively by established manpower program methods. They seek, through innovative techniques and new types of organizational arrangements, to determine how the programs might better "reach" and help prepare such workers for jobs, place them, and retain and upgrade them in gainful employment.

Because each project is specially designed, experimental and demonstration projects are not readily categorized. They differ widely, not alone by group or problem focused upon, but by technique or combination of techniques tried and, of great importance, by type of institution or combination of institutions enlisted to conduct the effort.

The groups concentrated on have been primarily unemployed ghetto area youth, minorities with cultural, emotional and other handicaps to employment, low-income rual residents, and older workers with limited education.

Although the E and D program's key objective is to stimulate and guide innovation rather than to provide services directly, it does provide significant assistance to the thousands of participants in its projects.

Many of the techniques for delivery of manpower services have been developed or refined in E and D-sponsored projects. Briefly, important concepts which E and D efforts have helped pioneer and introduce widely into manpower programming include:

(a) outreach to identify, attract and retain participation of the disadvantaged who do not come forward on their own for needed manpower services; (b) multi-service programs and centers to provide comprehensive service on a coordinated readily-accessible basis; (c) work sampling to evaluate the potential of those with limited education and to build the confidence of those with limited communication skills; (d) prevocational training, work orientation and related preparation as an aid to effective skill training and employment; (e) use of nonprofessional and indigenous staff as a vital aid in manpower development for the disadvantaged; (f) new occupations, particularly as subprofessional aides in human service activities, to broaden opportunity for the undereducated; (g) use of community and minority organization capabilities to complement government agency manpower development efforts; (h) inducements for employer initiative and action to hire, orient, train, and retain workers customarily regarded as "unacceptable"; (i) post-placement coaching and "high support" to enable employers and disadvantaged workers to overcome difficulties jeopardizing job retention in the initial months after hiring.

More specifically, the following are major examples of types of E and D accomplishments:

The major new Concentrated Employment Program (CEP) and Job Opportunities in the Business Sector (JOBS) manpower programs, initiated in part on the basis of E and D findings, were given significant start-up assistance by the E and D program:

1. Many features of the CEP have been designed from examples developed by E and D projects. The orientation, coaching, and employer involvement components particularly are based on E and D-developed models. Several E and D projects, most notably the JOBS NOW program in Chicago, provided the initial staff training and technical guidance for CEP personnel. And key staff needs in several of the initial CEPs were filled by personnel drawn from E and D projects. The E and D program also developed specific guide materials on job development methods, orientation and coaching to assist the new CEPs in such activities.
2. The new JOBS program initiated with the National Alliance for Businessmen was similarly influenced by E and D pilot experience. The findings of several E and D projects shaped the guidelines for JOBS efforts, and materials developed in the E and D program have served as basic resources for JOBS employer-contractors.

New ways have been developed by E and D projects to open and improve employment opportunities for the disadvantaged in major occupations:

1. E and D projects in Cincinnati and Washington have with union cooperation been exploring how to provide work preparation and experience for disadvantaged, particularly minority, youth to enable them to enter building trades apprenticeships and employment in housing renovation and construction. These projects have been looked to as practical examples to aid development of Model City program guidelines for employment of neighborhood residents in ghetto rebuilding.
2. A demonstration project with the Post Office Department has developed a technique which other Federal agencies are considering to help overcome test barriers to employment of the disadvantaged. Workers unable to pass civil service tests were recruited and hired on a temporary basis and, after special instruction while employed, a high proportion were enabled to meet the test requirement for permanent employment—and have performed effectively on the job.

Techniques are being developed to help employers upgrade their unskilled workers. A pilot E and D effort provided brief but intensive in-plant training to workers in traditionally dead-end jobs to qualify them for upgrading to newly designed higher-level jobs which the employer might not otherwise fill from his own employees. The employer response to this project has led to its extension for further development in new projects in three major cities, preparatory to likely larger-scale application in the near future.

Techniques are being developed to help identify the "real" job potentials of disadvantaged persons. The disadvantaged person's lack of skills and insufficient knowledge of his own capabilities combined with his usual very poor performance on paper-and-pencil tests all conspire to qualify him in the eyes of the counselor or personnel man for only the most menial dead-end jobs.

Work sample tests, originally developed by sheltered workshops for physically and mentally disabled, have been shown by E and D to be useful with the disadvantaged as a substitute for the unworkable written tests. The work-sample technique has been refined by the Philadelphia Jewish Employment and Vocational Service in an E and D project that has led to a ten-city pilot operation that will further extend our knowledge of its utility as an approach to appraising the job potentials of the unemployed.

Other interesting E and D efforts in their early stages are:

1. *Crime problems.* E and D efforts on several projects are designing systems with courts and police to develop training and employment as an alternative to criminal prosecution and imprison ment.
2. *Job language facility.* Projects are focusing, not on basic literacy as such, but on "job English" for Spanish speaking workers and on "occupational language" for workers with limited literacy backgrounds.
3. *Employer Based Day Care.* One project is exploring the feasibility and value of an employer sponsored day care center as an aid in recruiting the inner-city unemployed for existing job vacancies, and as a means of enhancing employee job stability and performance.

The E and D program's emphases will steadily shift as earlier findings are absorbed by established programs and attention is required by emerging new manpower problems and by a growing need for measurement and analysis of relative effectiveness of alternative approaches.

A panel session—Computers in service to libraries of the future

CALVIN N. MOOERS, *Chairman of Session*
Rockford Research Institute Incorporated
Cambridge, Massachusetts

Computers in service to libraries of the future: Library requirements

by W. N. LOCKE
Massachusetts Institute of Technology
Cambridge, Massachusetts

An outstanding computer engineer recently compared libraries to the whaling industry, a relic of the romantic past. As whales disappeared, so will books, he said. We should stop building libraries, store all information on tape and retrieve it through consoles.

This may be acceptable as a piece of blue skying but hardly comes to grips with the problems of information handling today and tomorrow. At a time when the world outpouring of written words is going up ten percent a year (an estimated 300,000 books and 100,000 serial titles in 1968), it doesn't make much sense for librarians or anybody else to plan in terms of a replacement of print by any other form in the near future.

So let's come down out of orbit and talk about mundane facts. Libraries cost dollars and serve people. For dollars they compete with other goods and services. The people they serve are as diverse as the population. Some just want a quiet, comfortable place to read or think; others want a particular book or journal; still others want all the information you have on some special topic; some need items that have to be located and brought from some other library. Then there are those who want to check a fact, a name, and so on. The library has to be all things to all people. And this requires complex organization, specialized staff, and constantly expanding space: it requires a lot better inventory control techniques than we now have. This is the challenge to computerniks. If they want to take over and operate the information handling business, they must do so in a real world of program budgets and cost benefit analysis. They must also work closely with librarians to provide a transition from the present to the future.

It might be well to look at today's information retrieval in the library context, see how much of it goes on, and calculate the cost. This amount is in the budget and presumably available for a computerized service. The total is not encouraging. Additional services that can be provided by the computer will have to be costed out and budgeted for next year or some future year.

At present, it is the customer who does most of the information retrieval; only he, and frequently not even he, knows what he wants. The library staff spends most of its time on document handling, acquiring, cataloging, and retrieving, not information, but books and book-like materials in dozens of forms, full size and mini, plus maps, music scores, manuscripts, sound and video recordings. The library is the memory of the race. It is different from the memory of the individual in that the individual's memory is associative while the library deals with discrete packages. Cataloging is a poor but expensive substitute for what goes on automatically and subconsciously in our minds as we record our experience.

About 33 percent of the average library personnel budget goes into the preparation and filing of information about the many kinds of items that come into the library. This input can as well go into a computerized as into a manual system, but it is hard to see how savings can be effectuated by a computer at this point unless we can get machine readable input ready-made from a source like MARC II tapes. If it is cheaper to process these tapes to find the descriptive and subject catalog information for individual items as they come to a library, rather than get it from printed copy or originate it, then we will use them. Unfortunately, present indications are that it may cost more.

Storage is one of the cheapest things we have today. Even amortizing land and building cost, we can keep reference books a year on the shelf for 20¢ apiece. At 5 M bits each, this is 4¢ per million bits per year. The card catalog is somewhat more expensive at 48¢ per year per million bits. Add an annual increment of eight percent or so and costs of conventional storage are still bearable. Not so the costs of on-line storage, which is the only conceivable form of computer storage for this type of material.

The numbers are not much better for present day library reference staff work in information retrieval. Reference work and catalog service may account for about ten percent of the personnel budget of a large library. Half of the time of these professionals may go into information retrieval. Assuming that they will have to spend a good deal of time training customers in using the computer, we may be able to save half the present budget to put toward the machine. This will not go far. In fact, my conclusion is that computerized information retrieval will require practically all new money. Major new financial support will be needed for large scale information retrieval, SDI and other individualized computer based services which we in the libraries want to provide.

Librarians have always been quick to adopt new technology, for instance for catalog card production, for micro storage, for quick, expendable copies. Computers are no exception. They are urgently needed now for inventory control. If we can afford anything that computers have to offer, it is this.

Using computer technology—Frustrations abound

by HENRIETTE D. AVRAM

Library of Congress
Washington, D. C.

The automation of libraries is a fairly recent entry to the growing number of areas of applications for computers. Is this an indication that librarians have been resisting advancing technology or could it be that the process of controlling large stores of information is so complex and the hardware, software, and brainware still too limited to cope with this complexity? Might it also be that computer specialists, underestimating the challenges, have evinced little interest in the library problem?

My experience in the library world suggests that these states and conditions have all combined with negative effect. The function of a library is to provide reference service to users and to make readily available the contents of its collections. The efficient performance of this function is directly related to the successful and timely completion of processing, i.e., the selection acquisition, cataloging, classification, and shelving of a book. The rapidly increasing number of books and periodicals places the greatest strain in this area and thus pinpoints the prime candidate for mechanization.

Before discussing one of the major automation activities at the Library of Congress and its associated problems, some facts about LC are in order to set the environmental background. The Library of Congress has in its collections about 55.5 million items: books, serials, maps, music, prints and photographs, manuscripts, etc. Approximately 75 million records contain the control information and bibliographic description of this collection. Its largest file, the Official Catalog, contains some 14.5 million records. An inventory of files showed that there are about 1,260 different files which are used in the Library's operations. Under Title II-C of the Higher Education Act of 1965, the Library has been charged with the additional responsibility of acquiring and cataloging all works, published anywhere in the world, important to scholarship. The materials flowing into the Library include items written in 70 different languages, represented by 20 distinct alphabets.

One of the basic functions of librarians is the recording and organizing of bibliographic data to facilitate access to and use of the books and other materials contained in the collections of libraries. Although bibliographic data may be recorded and stored in a variety of ways, the card catalog record has been the preponderant medium used by libraries in the United States. The bibliographic information on the catalog record is basically of two kinds: (1) a description of a book in terms of author, title, etc., and (2) some kind of notation to be used in locating the book on the shelves. The locating notation also usually comprises a means for arranging together materials on the same and related subjects. A catalog record distinguishes in a unique place one book from all the other books represented in the catalog. The catalog card, with its basic information, can be used again and again to provide multiple access capability—usually author, title, subject—and forms the basis of what is known as the unit card system. Essentially librarians are attempting to organize and make readily available the intellectual output (books) of other humans in all disciplines. This involves the application of a rather complicated set of rules to the output of very unpredictable human beings, the result being that it is safe to say that almost every rule will find its exception manifested.

Since the Library of Congress is the major source of bibliographic information for the American library community, it was natural to conduct an experiment at LC to test the feasibility and utility of centrally producing cataloging data and distributing these data to users. Project MARC (for MAchine-Readable Cataloging) was in operation for 19 months in test and pilot phases involving sixteen cooperating libraries. The project was successful and a full operational system providing selected machine-readable cataloging data for all interested libraries will begin early in 1969. During the pilot period, recommendations for improvement were received from the participants, a cost model was maintained, and the procedures for preparing bibliographic data for conversion to machine-readable form and the processing of these data were improved. The format for the interchange of the record was evaluated by staff members of many organizations: the Library of Congress, the National Library of Medicine, the National Agricultural Library, the United States of America Standards Institute Z39 Subcommittee 2 on Machine Input Records, the Committee on Scientific and Technical Information (COSATI), and other interested organizations both here and abroad. The result was the adoption of a format designed for the interchange of data and hospitable to the bibliographic description of all forms of material.

The format for monographs as adopted by the Library of Congress has four important characteristics:

1. It establishes the means by which bibliographic information may be transmitted between libraries.
2. It describes the rigorous rules by which bibliographic information, available in human-readable form, may be converted to machine-readable form.
3. It suggests that if the same format is used for the exchange of information by all libraries, programs and procedures may be exchanged and automation costs reduced.
4. It follows the United States of America Standards Institute Code for Information Interchange (ASCII), the standard for Recorded Magnetic Tape for Information Interchange, and the proposed standard for Magnetic Tape Labels and File Structure.

The library community, although operating in a very imperfect world in terms of having both second and third generation computers, configurations progressing from minimal to maximum (when is a 1401 a 1401?), and I/O devices not capable of handling the necessary character sets, has forged ahead to adopt standards. This is a significant step forward.

The introduction of computers to libraries poses special problems in file organization and hardware while providing new opportunities for multiple access to information. We are faced with deciding how information can best be structured and stored for effective retrieval. Imposed on top of all classic functions performed by librarians, i.e., acquisitions, cataloging, classification, reference, is the function of searching. The search argument varies with the inquiry. It ranges from data on an order slip to the information on the title page of a book, to the Library of Congress catalog card number, to a name in an authority file. The questions of file structure—where in the file to search and when to stop searching—are related to discovering the criteria for the determination of identity. The human mind has certain categories of analytic capability which cannot yet, if ever, be captured by machine. Therefore, we must create ploys which cause the machine to approach, in effect, the desired objectives.

Studies at the Library of Congress show that the storage requirements for 1972 is 4×10^9 characters. Interesting developments in hardware technology in the next five years should partially resolve the problems of large random access stores at acceptable costs. If we can approach an efficient solution for organizing information and consequently retrieving from the files, one nagging question that remains is how best to convert the files and in what order of priority. Because libraries cannot limit coverage in time and discipline, files reflecting the past must in time be converted to machine-readable form. Many conversion strategies have been proposed and the final decision must be based upon reasonable grounds as to use and cost. The conversion of bibliographic information requires specifications for the representation of this information in machine-readable form, i.e., decisions regarding data elements that need explicit identification and the definition of a character set for input, storage, and display. The character set needed to encode bibliographic data is essentially infinite because it is open-ended. Not only are we concerned with many languages in a multiplicity of alphabets, but in addition, any author can use any character at will. The obstacles then become challenges seeking creative solutions.

Librarians and computer scientists have rarely communicated well with one another, and this lack of communication results from the fact that each group is too parochially oriented to its own field. Both groups are actually striving toward precision but each sees precision in a different way. The librarian is concerned with precision in the definition of the record, for he must be precise in this definition in order to uniquely represent a book for retrieval. The computer person is interested in precision in method, i.e., an exact description of a process, so that his program will perform efficiently and produce the output required. Machine people have a tendency to minimize the librarian's problems of precision and exhibit a general reluctance to become interested in the data except as it affects the computer application. Without a complete understanding of the complexity of the data, the capabilities of the computer are oversold, thus later causing what might be termed a credibility gap. Librarians, on the other hand, must recognize the potential and the limitations of the new technology and provide the necessary guidance for the efficient use of communication and information manipulation devices.

Success will not come overnight but will depend upon the combined efforts of the most talented people that can be found in many disciplines.

Computers in service to libraries of the future

by HOWARD W. DILLON

Harvard University
Cambridge, Massachusetts

Automation activities in libraries have been undertaken with accelerating frequency over the past ten years. It is no longer uncommon to find successful projects in almost every type and size of library throughout the country. Libraries have demonstrated that they can develop and operate ordering and processing systems to control financial and bibliographic information at the time a new item is added to the collection. Book catalogs and other holdings lists are produced and distributed in many formats. Automated circulation control systems, particularly the data collection type, are widely accepted and functioning successfully.

This portion of the panel discussion will describe a few projects with which the speaker is familiar. The projects selected have been chosen because they represent major attacks on problems which must be solved in order for all libraries to move forward with automation.

A recurring problem for systems of library processing developed over the past years has been the reliability of the data first entered into the system at the time a purchase order is placed. In off-line systems, the need to edit or replace the information used at the time of the order with better data obtained when the item purchased is in hand has been a critical update problem. Few systems, therefore, attempted to carry computer processing from ordering through to the completion of cataloging as one integrated system. On-line processing capabilities and the development of a nationally distributed, timely bibliographic record by the Library of Congress in the MARC II communications format, make it more likely that integrated technical processing systems for libraries can be made operational.

Second, given a communications format for the sharing of bibliographic data, libraries with common processing requirements are undertaking to share in the cooperative development and design of processing systems. Standardization and compatibility have been hallmarks of library systems for many years. While not always perfectly achieved, librarians have traditionally demonstrated their concern with the sharing of bibliographic data and collections. The advent of computer processing has not altered that basic philosophy. Rather, it provided an opportunity to realize the goal of compatibility with greater perfection.

Systems to be considered in this presentation will include:

SYMBIOSIS—**SY**stem for **M**edical and **BIO**logical **S**ciences **I**nformation **S**earching

NELINET —The **N**ew **E**ngland **L**ibrary **I**nformation **Net**work

The Integrated, Computer-Based, Bibliographical Data System for a Large University Library, being developed by the University of Chicago Library, including the subsequently coordinated activities of Columbia and Stanford university libraries in this acquisitions and cataloging system.

The Washington State University Library Technical Services System, which is a project to develop an on-line processing capability for that library.

The presentation will review the objectives of these projects, summarize their accomplishments to date, and discuss hardware or operating system software problems encountered.

Batch, conversational, and incremental compilers

by HARRY KATZAN, JR.

Pratt Institute
Brooklyn, New York

INTRODUCTION

Compiler-writing techniques have received a great deal of pragmatic and academic attention and are now fairly well-defined.* It was and still is generally felt that the compiler is independent of the operating system in which it resides, if it resides in one at all. The invention of time-sharing systems with conversational capability, however, has required that compiler experts re-evaluate existing concepts to make better use of external facilities. This was done and conversational and incremental compilers have evolved. A generalized and consolidated discussion of these relatively new concepts is the subject of this paper. First, a model of a batch compiler is introduced. The concepts are then modified and extended for a conversational programming environment. Finally, a recent development termed "incremental" compilation, which satisfies the needs of both batch and conversational compiling as well as interactive computing, is presented. First, some introductory material is required.

Basic concepts

In the classical data processing environment,** the "compile phase" or "source language processing phase" is of prime importance as are definitions of *source program* and *object program*. The latter are redefined in light of the time-sharing or interactive environment. Extraneous items, such as where the object program is stored or whether or not the compiler should produce assembler language coding, are practically ignored.

The *source program* is the program as written by the programmer. It is coded in symbolic form and punched on cards or typed in at the terminal. The *object program* is the program after being transformed by the compiler into a machine-oriented form which can be read into the computer and executed with very few (if any) modifications. Also of interest is the *information vector* which gives initial conditions for compilation and denotes the types of output desired. A sample of specifications which might be found in an information vector follow: (1) location of the source program; (2) name of the program; (3) the extent of compiler processing, i.e., syntax check only, optimize, etc.; (4) computer system parameters; (5) compiler output desired; and (6) disposition of the object module. The form of the source program is sometimes required, although in most cases this information is known implicitly. This pertains to different BCD codes and file types which may range from sequential or indexed files on conventional systems to list-structured files in virtual machines.

Similarly for output, the user can request a specialized form of object module or none at all, source or object program listing, and cross-reference listings. The object module is known as a Program Module which contains the machine language text and relocation information. Additionally, it may contain an Internal Symbol Dictionary for use during execution-time debugging. The Internal Symbol Dictionary is especially useful in conversational time-sharing systems where execution can be stopped on a conditional basis and the values of internal variables can be displayed or modified.

Batch compilation

Batch compilation methods are required, quite naturally, in a batch processing environment. The term "batch processing" stems from the days when the programmer submitted his job to the computer center

* Two books devoted entirely to the subject are worth mentioning: Lee, J.A.N., *The Anatomy of a Compiler*,[1] and Randell, B. and L. J. Russell, *Algol 60 Implementation*.[2]

** See Lee,[1] p. 9.

and subsequently received his results later in time. A collection of different jobs was accumulated by operations personnel and the batch was then presented to the computer system on an input tape. The important point is that the programmer has no contact with his job between the time it is submitted to operations and when he receives his output. The concept has been extended to cover Multiprogramming Systems, Remote Job Entry (RJE), and the trivial case where no operating system exists and the programmer runs the compiler to completion.

The generalized batch environment

The most significant aspect of the batch processing environment is that the entire source program is available to the compiler initially and that all compiler output can be postponed until a later phase. The compiler writer, therefore, is provided with a liberal amount of flexibility in designing his language processor. For example, specification (i.e., declarative) statements can be recognized and processed in an initial phase and storage allocated immediately. In the same pass, statement labels are recognized and entabled; then in a later phase, validity decisions for statements that use statement labels can be made immediately rather than making a later analysis on the basis of table entries. If desired, source program error diagnostics can be postponed. Moreover, the designer may specify his compiler so that the source program is passed by the compiler or so that the compiler is passed over the source program, which resides semi-permanently in memory.

This inherent flexibility is not exploited in the compiler model which follows. Instead, an attempt has been made to present the material in a conceptually straightforward manner.

A generalized batch compiler

By itself, a model of a generalized batch compiler is of limited interest. The concept is useful, however, for comparison with those designed to operate in time-shared computer systems. Therefore, the presentation is pedagogical in nature as compared to one which might present a step by step procedure for building one.

Processing by the compiler is rather naturally divided into several phases which tend to be more logical than physical. Each phase has one or more specific tasks to perform. In so doing, it operates on tables and lists possibly modifying them and producing new ones. One phase, of course, works on the source program from the system input device or external storage and another produces the required output. The entire compiler is described therefore by listing the tasks each phase is to perform; ordinarily, the description would also denote which tables and lists each phase uses and what tables and lists it creates or modifies. The specific tables and lists which are required, however, tend to be language dependent and are beyond the scope of this treatment.

The compiler is composed of five phases and an executive routine, as follows:

The Compiler Executive (*EXEC*). The various phases run under the control of a compiler executive routine (EXEC) which is the only communication with the outside world. It establishes initial conditions and calls the different phases as required. It can be assumed that EXEC performs all system input/output services, upon demand from the phase modules. More specifically, the EXEC has five major and distinct functions:

1. to interface with the compiler's environment;
2. to prepare the source statements for processing by phase one;
3. to control and order the operation of the phases;
4. to prepare edited lines for output; and
5. to provide compiler diagnostic information.

Phase 1. Phase 1 performs the source program syntactic analysis, error analysis, and translation of the program into a tabular representation. Each variable or constant is given an entry in the symbol table, with formal arguments being flagged as such. Initial values and array dimensions are stored in a table of preset data.

Lastly, information from specification statements is stored in the specification table. The most significant processing, however, occurs with respect to the *Program Reference File* and the *Expression Reference File*.

Each executable statement and statement label is placed in the Program Reference File in skeletal form. In addition to standard Program Reference File entries, the Program Reference File contains pointers to the Expression Reference File for statements involving arithmetic or logical expressions.

The Expression Reference File stores expressions in an internal notation using pointers to the symbol table when necessary. As with the Expression Reference File, the Program Reference File also contains pointers to the symbol table.

Phase 2. In general, phase 2 performs analyses that cannot be performed in phase 1. It makes storage assignments in the Program Module for all variables that are not formal parameters. It detects illegal flow in loops and recognizes early exits therefrom. It also determines blocks of a program with no path of control to them; and lastly, it detects statement labels which are referenced but not defined.

Phase 3. The object of phase 3 is to perform the global optimizations used during object code generation, which is accomplished in phase 4.

The first major function of phase 3 is the recognition and processing of *common sub-expressions.* Phase 3 determines which arithmetic expressions need be computed only once and then saved for later use. In addition, it determines the range of statements over which expressions are not redefined by the definition of one or more of their constituents. If the occurrence of an expression in that range is contained in one or more DO* loops which are also entirely contained in that range, Phase 3 determines the outermost such loop outside which such an expression may be computed, and physically moves the expression to the front of that DO loop. Only the evaluation process is removed from the loop; any statement label or replacement operation is retained in its original position. The moved expression is linked to a place reserved for that purpose in the program reference file entries corresponding to the beginning of the respective DO loops.

The second major function of phase 3 is the recognition and processing of *removable statements.* A "removable statement" is one whose individual operands do not have "definition points" inside the loop; obviously, the execution of this statement for each iteration would be unnecessary. A definition point is a statement in which the variable has, or may have, a new variable stored in it (e.g., appears on the left-hand side of an equal sign). In removing statements, they are usually placed before the DO statement.

Phase 3 also processes formal parameters and develops the prologue to the program; it optimizes the use of registers; and it merges the Program Reference File and the Expression Reference File to form a Complete Program File in preparation for phase 4.

Phase 4. Phase 4 performs the code generation function. Its input consists of the symbol table and the Complete Program File and its output is the Code File, which represents completed machine instructions and control information.

Phase 5. Phase 5 is the output phase and generates the *Program Module,* the source and object listings, and the cross reference listing. Upon request, an *Internal Symbol Dictionary* is also included in the Program Module.

* Although the DO keyword is a constituent part of several programming languages, it should be interpreted as representing the class of statements from different languages which effectively enable the programmer to write program loops in a straightforward manner.

Any compiler model of this type is clearly an abstraction; moreover, there is almost as much variation between different compilers for the same programming language as there is between compilers for different languages. The model does serve a useful purpose, which is to present a conceptual foundation from which conversational and incremental compilers can be introduced.

Conversational compilation

Compared with the "batch" environment in which user has no contact with his job once it is submitted, the conversational environment provides the exact opposite. A general-purpose time-sharing system of one kind or another is assumed,* with users having access to the computer system via terminal devices.

In the batch environment, the user was required to make successive runs on the system to eliminate syntax and setup errors with the intervening time ranging from minutes to days. Excluding execution-time 'bugs", it often took weeks to get a program running. In the conversational mode, syntactical and setup errors can be eliminated in one terminal session. Similarly, execution-time debugging is also possible in a time-sharing system, on a dynamic basis.

Conversational programming places a heavy load on a compiler and an operating system; the magnitude of the load is reflected in the basic additions necessary to support the conversational environment.

The time-sharing environment

The time-sharing environment is characterized by versatility. Tasks can exist in the "batch" or "conversational" mode. Furthermore, source program input can reside on the system input device or be prestored. The time-sharing operating system is able to distinguish between batch and conversational tasks; therefore, batch tasks are recognized as such and processed as in any operating system. The ensuing discussion will concern conversational tasks. It is assumed, also, that the user resides at a terminal and is able to respond to requests by the system.

During the compile phase, the source program may be entered on a statement-by-statement basis or be prestored. In either case, the compiler responds immediately to the terminal with local syntactic errors. The user, therefore, is able to make changes to the source program immediately. Changes to the source program other than in response to immediate diagnos-

* Two typical general-purpose time-sharing systems are TSS/360[3,4] and MULTICS.[5]

tics cause a restart of the compilation process. Obviously, the system must keep a fresh copy of the source program for the restart case. To satisfy this need, a copy of the current up-to-date source program is maintained on external storage; if the source was prestored, the original version is updated with change requests; if the source program is not prestored, the compiler saves all source (and changes) as they are entered line-by-line. With the user at a terminal, the compiler is also able to stop midway during compilation (usually after the global statement analysis and before optimization) to inquire whether or not the user wants to continue. Under error conditions, the user may abort the compilation or make changes and restart the compilation process. Moreover, the user can utilize this pause to have his program syntax checked only.

During the execution phase, dynamic debugging is often desirable. This facility is usually a part of the command structure of the operating system. In preparation for execution-time debugging, the user would probably request an Internal Symbol Dictionary during compilation so that internal variables can be addressed symbolically. Since execution-time debugging is a relatively new concept, it is discussed briefly.

Debugging commands usually fall into three categories: (1) program control; (2) program modification; and (3) debugging output. Debugging commands may be imbedded in the program itself or the program can be stopped (either asynchronously or with an AT command) and the actions performed immediately. Examples of typical *program control* commands are:

```
AT     symbolic-location . . . STOP
RUN
RUN    symbolic-location
```

Examples of *program modification* commands are:

```
SET  A = 1.0
IF A <0,  SET  A = 0
```

Examples of *debugging output* commands are:

```
DISPLAY MAIN.I:MAIN.A
DUMP ARRAY
```

Furthermore, they can be used in combination as follows:

```
AT  PTWO.100     IF A = 0,     STOP
AT T34.360 DUMP T34.A SET CNT = CNT + 1
```

As was mentioned earlier, a considerable amount of the compiler's effort is devoted to producing an efficient object program. As a result, the instructions to perform certain computations are sometimes not located where one would expect to find them. In fact, this is a direct consequence of *common sub-expressions* and *removable statements*, which were discussed previously. Although these processes contribute to efficiency, they have a side effect which hinders the debugging effort. Therefore, when expecting to use dynamic debugging, the user should request an Internal Symbol Dictionary and select the option which does not produce optimized code.

The conversational compiler and the time-sharing operating system must support several aspects of the conversational environment. These are summarized as follows: (1) the ability to change or forget the effects of the preceding statement; (2) restart logic; (3) maintenance of the entire source program, in up-to-date form, on external storage; (4) the ability to scan statements and produce diagnostics on an individual statement basis; and (5) the option to produce optimized or unoptimized code.

The conversational compiler

Basically, the conversational compiler is a conventional batch-processor containing special features making it suitable for conversational, terminal-oriented operation.

Structurally, the major addition over a batch compiler is Compiler Control Program (CCP), which in effect controls compilation. CCP is cognizant of whether the mode of operation is batch or conversational and is able to fetch source records and dispose of output print lines, accordingly. CCP is the facility which maintains the source program on external storage and is able to tell if a new source record is indeed new, a change to last one entered, or a change to a previous one. When processing a request to fetch a source record for the compiler, CCP can use this information to simply return the record, return it with the "forget" flag on, or call the compiler at its initial entry for the restart case. The function to fetch a source record is termed GET-LINE and is summarized in Table I. Accordingly, an overview of the CCP is given in Figure 1.

The overall logic of the conversational compiler is shown in Figure 2. Clearly, it differs very little from the batch version. The differences in the compiler itself are found in phase one and at the end of phase two. In phase one, as shown in Figure 3, the compiler uses CCP as its external interface. Moreover, the compiler always compiles a statement conditionally; later it uses the

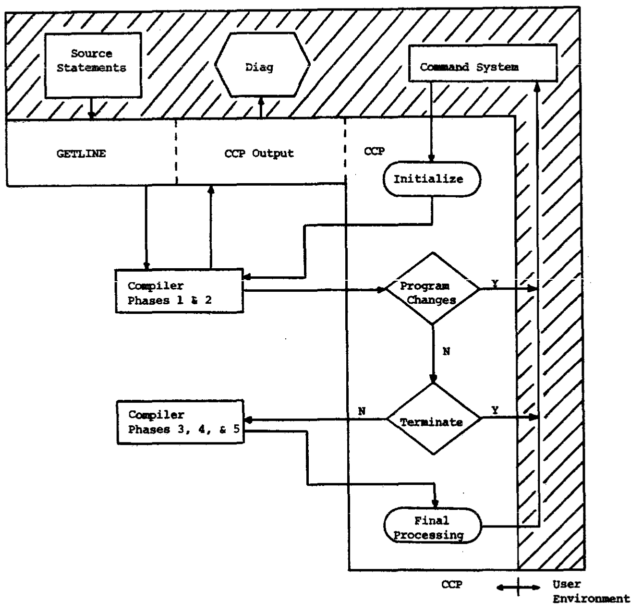


Figure 1—Overview of the compiler control program (CCP)

Table I—GETLINE Function of the Compiler Control Program (CCP)

	Conversational		Batch	
	Prestored	Not Prest.	Prestored	Not Prest.
GETLINE	A	B	A	C

A. Fetches the next source record from external storage and returns it to compiler EXEC.

B. Fetches another source record from the terminal input device and updates the source file on external storage. If it is the next source record, the line is returned to the compiler with the "forget" flag off. If the given source record is to replace the previous one, the "forget" flag is turned on and the line is again returned. Otherwise, a previous line has been modified and the compiler is entered at the "initial" entry point for the restart case.

C. Fetches the next source record from the system input device and updates the source file on external storage; the line is returned to EXEC with the "forget" flag off.

"forget flag" to freeze or delete the compiled information.

After phase two, as shown in Figures 1 and 2, the conversational compiler again exits to CCP. In the batch mode, of course, CCP simply returns to the compiler. In the conversational mode, as shown in Figure 1, the user is asked for changes and whether he wants to

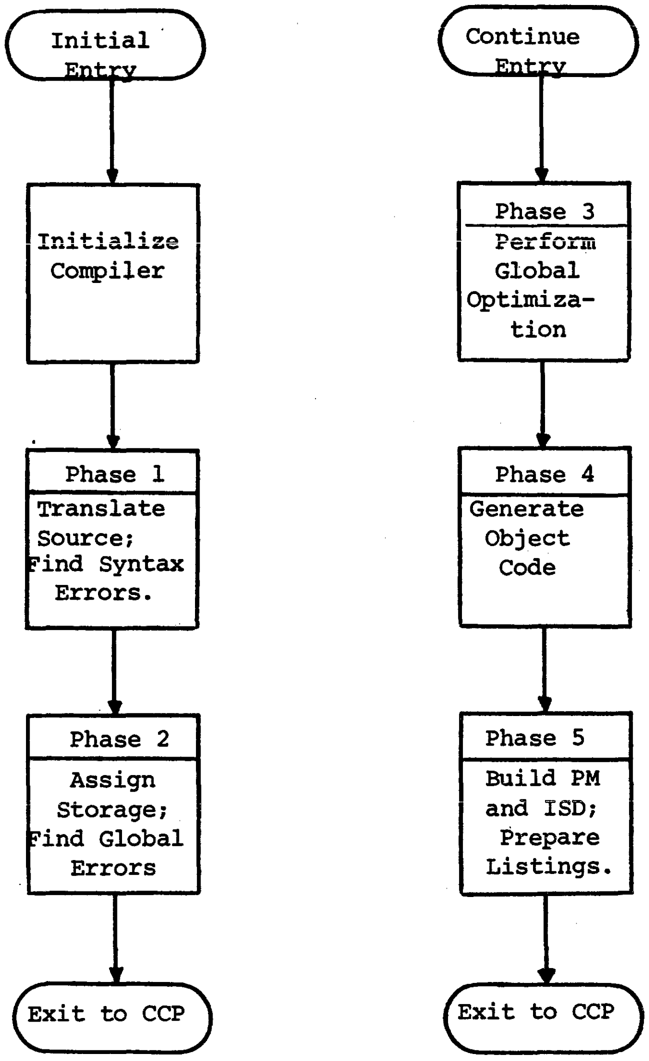

Initial Entry
Initialize Compiler
Phase 1
Translate Source; Find Syntax Errors.
Phase 2
Assign Storage; Find Global Errors
Exit to CCP
Continue Entry
Phase 3
Perform Global Optimization
Phase 4
Generate Object Code
Phase 5
Build PM and ISD; Prepare Listings.
Exit to CCP

Figure 2—Logic of the conversational compiler

continue. At the user's request, CCP can change the source program, still residing on external storage, and restart the compiler at the "initial" entry. If the user desires to continue, the compiler is entered at the "continue" entry. Otherwise, CCP exits to the command system and the remainder of the compilation is aborted.

Conversational compilation offers significant advantages over standard batch processing, most of which deal with the interactive mode of operation. The major disadvantage is that the entire source program must be available before execution can be attempted. In other words, one would like the versatility and flexibility of a language interpreter with the performance of a conversational or batch processor. Moreover, the performance must be reflected in the execution time as well as the compile time.

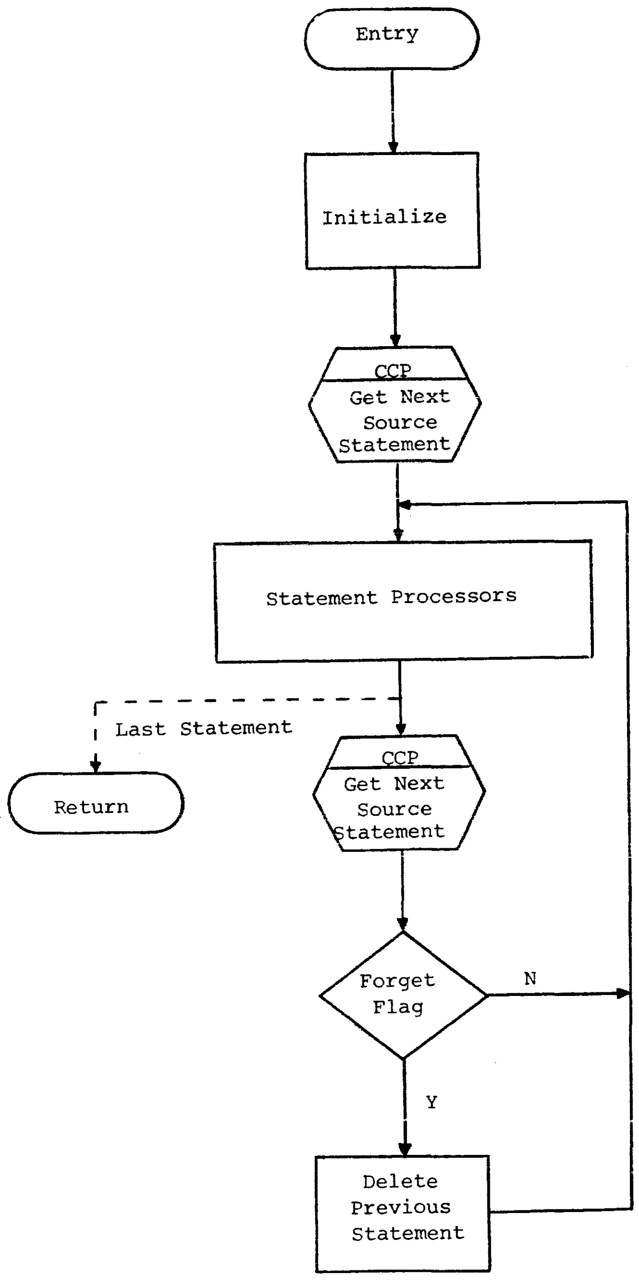

Entry
Initialize
CCP
Get Next Source Statement
Statement Processors
Last Statement
Return
CCP
Get Next Source Statement
Forget Flag
N
Y
Delete Previous Statement

Figure 3—Compiler Phase 1 interface with the compiler control program

Incremental compilation

One of the most promising ideas in this era of on-line computing is a concept termed Incremental Compilation. In an interactive programming environment, one would like to achieve both the speed factors inherent in compiled programs and the flexibility available with interpretive systems. Incremental compilers are an attempt to achieve these goals. Much of the pioneering work in this area is reported in two papers: the first by Lock[7] entitled "Structuring Programs for Multiprogram Time-Sharing On-Line Applications" and the second by Ryan[8] and others entitled "A Conversational System for Incremental Compilation and Execution in a Time-Sharing Environment." In order that the above goals can be realized, the following capabilities are required:

1. The ability to execute a statement immediately;
2. The ability to modify a prior statement without forcing a recompilation;
3. The ability to execute a source program as it is being input;
4. The ability to execute selected portions of programs;
5. A language processor that can also operate in the batch mode.

Clearly, all of the above requirements, except speed, are met with an appropriate interpretive program. In a large time-sharing environment, however, this resource is of prime importance, especially when 50 or more terminals are being serviced.

The environment for incremental compilation

Basically, the environment for incremental compilation is the same as for its conversational counterpart. By assuming a sophisticated operating system such as TSS/360[3,4] or MULTICS,[5] many of the problems described in the cited papers by Lock and by Ryan, such as memory protection among users, an effective command system, and memory organization and management, are obviated. Dynamic loading facilities, for utilizing hand coded subroutines, and a memory relocation feature,[9] for mapping virtual addresses to real addresses, simplify problems involving the execution of code compiled incrementally and are also assumed. The programming language is naturally of importance and of great interest to most systems programmers. A language more powerful than standard Fortran or assembler language is expected, although the techniques would work satisfactorily therewith. A rich language which would enable a significant amount of computation per interaction is most desirable. Languages such as PL/I[10] and Iverson's Language[11] are well suited to incremental compiling and executing.

The incremental compiler

This method of compilation permits two modes of operation: batch and incremental. In the *batch mode*, the user may compile prestored source program but may not modify or execute the program during compilation. In the *incremental mode*, normally used only conversationally, special facilities are available to permit the modification, execution, and checkout of the program during compilation. These operations are performed through a combination of control and source language statements.

Incremental compilation consists of accepting a source program on a statement by statement basis. Each statement is compiled as it is received and the code generated for it is immediately made available for execution. Associative links between the source program and object code are maintained, thus permitting the user, during compilation, to modify his source program and have the modification immediately reflected in the object code. The ability to compile, execute, and modify a program on a statement by statement basis gives the user a degree of flexibility over his program usually available only with an interpreter, yet reduces the principal objection to interpreters: that of requiring an excessive amount of execution time. While, in an interpreter, each statement must be processed each time it is executed, in an incremental compiler it needs to be processed only when it is entered initially or when the user makes a source program modification. The Incremental Compiler has the added advantage of ensuring that the object code the user tests incrementally is virtually the same as the code produced for an object module, since the same code generators are used in both modes.

When an Incremental Compiler is used in the batch mode, all of the usual facilities are available to the user. When used in the incremental mode, all of the batch facilities are available in addition to those provided to control the execution and debugging of the generated code. During both modes of compilation, the following options are permitted:

1. Analyze the program only for syntactic errors; do not perform a global analysis or generate code.
2. Analyze the program for syntactic and global errors; do not generate code.
3. Analyze the program for syntactic and global errors and generate object code as well.

The object program may be executed, in either the batch or incremental mode, only if the third option is selected. In most compilers of this type, the user may select one of several modes of executing the incremental code concurrently with compilation; as errors are uncovered, he may make modifications to the source language, without, in most cases, requiring a recompilation of the existing code or affecting previous execution.

In order to provide the user with the degree of control desired, two categories of *control statements* are necessary: transient statements and commands. A *transient statement* is a statement in the source language being compiled which is executed and discarded immediately. It allows the user to intervene during the execution of his program and print results or change values. *Commands* are control statements which allow the user to make modifications outside the scope of the source language. A good example would be to change the control point of the program.

Source program compilation and execution in the incremental mode is under direct control of a Language Controller (LC). Each interaction, by LC, with the user is divided into a processing cycle and/or an execution cycle, depending upon the input parameters. The compiler is called by LC to process source language and transient statements and if execution is requested, it is initiated and monitored accordingly. After execution, transient statements are discarded whereas source language statements are retained. The command system of the operating system is called by the Language-Controller to process commands. Clearly, there is a need to tie the various elements of a program together, for operational reasons, and this requirement is satisfied by the Program Structure Table, described below. Since the Language Controller controls a major portion of the processing when in the incremental modes, it is structured accordingly. As pictured in Figure 4, it contains program elements for maintaining the source program and the Program Structure Table, for controlling the compiler, for monitoring the execution of incremental code, and for interpreting and then dispatching or processing control statements. These functions are summarized in the following descriptions of the modules which comprise the Language Controller:

Program Structure Routines. The program structure routines maintain the source program on external

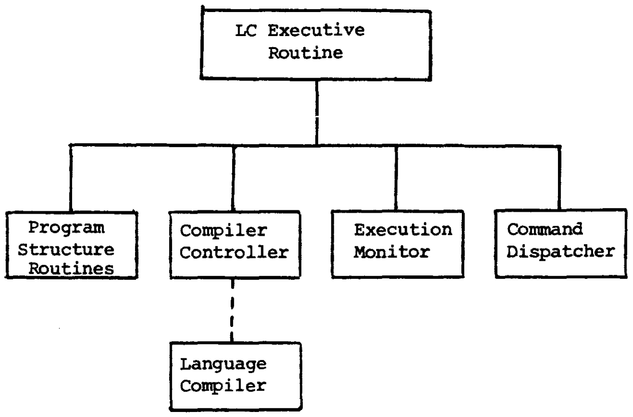


Figure 4—Structure of the language controller for incremental compilation and execution

storage and manage the Program Structure Table, which contains an entry for each source language statement in the program being compiled. The relationship of statements is also established for subsequent use by the Execution Monitor.

Compiler Controller. The compiler controller provides the interface between the user and the compiler. It passes the identity and location of source statements to the compiler EXEC and receives the location of the compiled code in return. In so doing, it handles diagnostics and updates the Program Structure Table.

Execution Monitor. The Execution Monitor controls program execution as determined by the established mode of operation. It passes control between statements or halts execution after specific statements, as required. It utilizes the dynamic loader of the operating system and envokes other program modules when requested to do so.

Command Interpreter and Dispatcher. The Command Interpreter analyzes control statements and calls either the Compiler Controller or the command system of the operating system depending upon whether a transient statement or a command is being processed.

The Program Structure Table is obviously of great importance since it indicates the relationship of statements and the static properties of the program. Elements in the table are generated dynamically as source language statements are entered and are composed from the following quantities:*

1. A *type indicator* specifying the type of statement;
2. A list of *structure pointers* linking this statement to preceding and succeeding statements and to any function module** in which it might be contained;
3. A pointer to the *compiled machine code* for the statement;
4. A locator, such as data set name or physical location, of the *source program* on external storage; and
5. A statement identification, such as a *line number*, used for referencing the statement and for making insertions, deletions, and changes to the program.

* The article by Lock contains a comprehensive description of internal program structure in a programming environment such as this.

**The term function module is used to represent either a *block* or internal *procedure* as found in Algol or PL/I.

Due to the nature of the incremental compilation process and the Program Structure Table, it is not necessary that the incremental code for a given program reside in contiguous memory locations. In fact, only rarely will this be the case. Although this is conceptually different from the established practice of generating object code, it poses no serious problem in the incremental mode of operation.

In general, the incremental compiler is composed of the same basic components as the batch and conversational versions. Some differences, which tend to be related to the interrelationship of statements, do exist but are relatively minor. The "global" analysis of statements, for example, is severly crippled by the fact that all statements in a source module may not be available for analysis. The "global" optimization of statements is in the same category but must be eliminated entirely. It is very feasible, however, to include it as a special phase in the batch mode or provide a mechanism to convert from incremental to object code, including global optimization, in the conversational mode.

The basic compiler processing cycle begins when it is called at its source input entry. (Another entry could conceivably exist which might be to convert incremental code to object code.) The compiler EXEC obtains the source text to be processed from the Language Controller and builds a Program Table consisting of the text to be processed during the cycle; information on additions, insertions, and deletions; location of the existing symbol table; and parameter data relating to mode of compilation, listing options, BCD codes, etc. The EXEC then invokes Phase 1 of the compiler which performs a statement classification and syntax analysis and builds the Program Reference File and Expression Reference File from all of the statements specified in the Program Table. Pointers to the encoded statements are then returned to the EXEC, where the encoded statements are linked back to the Program Table. Phase 2 is then invoked to perform a global analysis, when possible, and to assign storage for the statements indicated in the Program Table. This phase updates the symbol table and merges the Program Reference File and Expression Reference File to form the Complete Program File maintaining the links to the Program Table, as required. Phase 4† is now called to translate the encoded statements into object code forming the Code File. Phase 5 which must generate either object code or incremental code is considered below.

Operation of the compiler for each of the two basic modes, batch and incremental, can now be described. In a *batch compilation*, the source text available at entry

† Recognizing that no Phase 3 exists.

consists of the complete program. The Program Table passed to each component points to every statement in the source program, so that in a single cycle, the complete compilation is produced. Other than the Executive (EXEC), the only phase which must be aware of the batch parameter is Phase 5, which must build an object module instead of generating incremental code. Again, the object module consists of a program module (i.e., text and relocation data) and optionally, an Internal Symbol Dictionary. The text portion consists of the object code produced and is entirely self-contained, with the code generated for a statement linking directly to the code for the next statement. The source text available at entry to an *incremental compilation* may represent anything from a single statement to a complete program. Normally, however, the source text available represents only a portion of a program. The Program Table, therefore, contains a group of statements to be added or deleted in the current program. The Program Table is in the same form as for a batch compilation and does not require different handling by Phases 1, 2, and 4. In this mode, Phase 5 generates incremental code. Incremental code differs from an object module in that the Program Module (i.e., the relocation information) must be dynamically generated requiring some special processing by the Language Controller and the system's dynamic loader. The text is organized on a statement-by-statement basis with inter-statement linkage provided to allow the intervention by the Language Controller* at statement boundaries.

As a result of the incremental process, four modes of execution are possible: automatic, controlled, block step, and step. In the *automatic mode*, statements are executed by the Language Controller immediately after they are processed by the compiler. In the *controlled mode*, statements are executed only when explicitly requested by a RUN command, which may designate a range of statements. In the block step and step modes, an entire program (i.e., an external procedure) is available for execution. For the *block step* case, the Language Controller pauses for user intervention after each block (or possible subroutine) in the program. When the *step* mode is specified, Language Controller suspends object program execution after each statement.

Early efforts and special problem areas

The two specific references to incremental compilation, i.e., by Lock[7] and by Ryan,[8] are in a sense complementary. Lock tends to emphasize the structural aspects whereas Ryan emphasizes the systems aspects, even though different computers are involved.

The effort by Lock, and probably others, at the California Institute of Technology is part of an experimental time-sharing project for the IBM 7040 computer. The programming languages supported are ALGOL, FORTRAN, and LISP with much of the paper being devoted to the internal organization of programs in an on-line programming environment. The Conversational Compiler System, reported by Ryan and others, is an outgrowth of Lock's work and runs under an SDS 940 time-sharing system. ALGOL and FORTRAN are also supported here with the paper emphasizing the command language, memory organization, and compiling techniques.

These projects have uncovered some interesting problems of which the most significant, perhaps, is considered here. It involves changing a data declaration when executable code exists which uses the variables declared therein. In fact, the code may have been executed previously. Lock solved the problem by designing his pseudo-machine code so that all references to identifiers are indirectly addressed through non-relocatable entries in the user's symbol table. This certainly solves the problem but it partially nullifies one of the basic objectives of incremental compilation; that is, to gain the speed factor inherent in compiled code. A hardware mapping of identifiers to physical locations is feasible if relocation hardware and possibly a small associative memory are available, although it remains to be seen whether dynamic address translation can be used in this particular manner. Finally, one might ask the following philosophical question: "Is it unreasonable to require a recompilation following a change to a data declaration?" Clearly, the answer must be evaluated in light of the other benefits to be gained through incremental compilation.

CONCLUSIONS

The world of time-sharing and its potential for interactive computing at a general level has raised some interesting topics.

First, it should be recognized that although batch techniques are currently very efficient and well defined, they were developed of necessity. When these techniques gained their acceptance, the batch mode was the only operational procedure available for using the computer. The programming community should also recognize that program development in a batch environment may not be the most natural or the optimum method.

Second, it should be recognized further that conver-

* That is, the Execution Monitor.

sational techniques do not offer a complete solution in that execution of parts of a program is usually not permitted. Clearly, language syntax errors can be detected as they are being input and this is certainly a step in the right direction. But if a programmer has to develop his algorithm completely before any of it can be executed, he might as well compile in the batch mode and rely on execution-time debugging.

Some form of incremental compiling, therefore, seems to be the only answer in sight to questions regarding the development of algorithms in an interactive computing environment. The ability to execute a program as it is being compiled is certainly a natural way and very well may be optimum from a development point of view. It remains to be seen if the gains can justify the complexity of an incremental compiler.

REFERENCES

1 J A N LEE
The anatomy of a compiler
Reinhold Book Co New York 1967

2 B RANDELL L J RUSSEL
Algol 60 implementation
Academic Press New York 1964

3 W T COMFORT
A computing system design for user service
Proc F J C C 1965

4 C T GIBSON
Time-sharing in the IBM/360: Model 67
Proc S J C C 1966

5 F J CORBATO V A VYSSOTSKY
Introduction and overview of the MULTICS system
Proc F J C C 1965

6 IBM System/360 Time Sharing System
FORTRAN IV Program logic manual
IBM Corporation Y28–2019 Yorktown Heights N Y 1967

7 K LOCK
Structuring programs for multiprogram time-sharing on-line applications
Proc F J C C 1965

8 J L RYAN R L CRANDALL M C MEDWEDEFF
A conversational system for incremental compilation and execution in a time-sharing environment
Proc F J C C 1966

9 B RANDALL C J KUCHNER
Dynamic storage allocation systems
C A C M Vol 11 No 5 May 1968

10 G RADIN H P ROGOWAY
Highlights of a new programming language
C A C M Vol 8 No 1 January 1965

11 K E IVERSON
A programming language
John Wiley and Sons Inc New York 1964

TRANQUIL: A language for an array processing computer

by NORMA E. ABEL, PAUL P. BUDNIK, DAVID J. KUCK, YOICHI MURAOKA, ROBERT S. NORTHCOTE, and ROBERT B. WILHELMSON

University of Illinois at Urbana-Champaign
Urbana, Illinois

INTRODUCTION

TRANQUIL is the algorithmic language which will be used to write programs for ILLIAC IV, a parallel computer which has been described by Barnes *et al.*[1] ILLIAC IV is designed to be an array of 256 coupled processing elements (PE's) arranged in four quadrants in each of which the 64 PE's are driven by instructions emanating from a single control unit (CU). Each of the 256 PE's is to have 2048 words of 64 bit semiconductor memory with a 250 nanosecond cycle time and an instruction set which includes floating point arithmetic on both 64 bit and 32 bit operands with options for rounding and normalization, 8 bit byte operations, and a wide range of tests due to the use of addressable registers and a full set of comparisons. The PE's differ from conventional digital computers in two main ways. Firstly, each is capable of communicating data to its four neighboring PE's in the array by means of routing instructions. Secondly, each PE is able to set its own mode registers, thus effectively enabling or disabling itself for the transmission of data or the execution of instructions from its CU.

Figure 1 shows 64 PE's, each having three arithmetic registers (A, B, and C) and one protected addressable register (S). The registers, words, and paths in Figure 1 are all 64 bits wide, except the PE index registers (XR), mode registers, and as noted. The mode register may be regarded as one bit which may be used to block the participation of its PE in any action. The routing registers are shown connected to neighbors at distances ± 1 and ± 8; similar end around connections are provided at 1, 64, etc. Programs and data are stored in PE memory. Instructions are fetched by the CU in blocks of 8 words as required and are stored in a 64 word CU instruction buffer.

Figure 1 also shows the essential registers and paths in the CU and their relations to the PE's. Instructions are decoded and control signals are sent to the PE array from the control unit. Some instructions are executed directly in the CU; e.g., the loading of CU accumulator registers (CAR's) with program literals. Operand addresses may be indexed once in the CU and again separately in each PE. It is possible to load the local data buffer (64 words of 64 bits each) and CAR's from PE memory. Local data buffer registers and CAR's may be loaded from each other. A broadcast instruction allows the contents of a CAR to be transmitted simultaneously to all PE's. It is often convenient to manipulate all PE mode bits or a number from one PE in a CAR. For this purpose, the broadcast path is bidirectional.

The four control units may be operated independently, as pairs, or all together. In the united configuration, all 256 PE's are effectively driven by one CU and routing proceeds across PE boundaries. This allows some flexibility in fitting problems to the array.

If ILLIAC IV, or any other parallel array computer, is to be used effectively, it is essential that all possible parallelism be detected in those algorithms which are to be executed by that computer. This is difficult, if not impossible, if the algorithms are specified in languages such as FORTRAN and ALGOL which essentially express all computational processes in terms of serial logic, as required for conventional computers. Since it is also more convenient for the user to express array type computation processes in terms of arrays and parallel operations, rather than having to reduce the the inherent parallelism to serial computational form, the specification of a new language for array processor computation is clearly necessary.

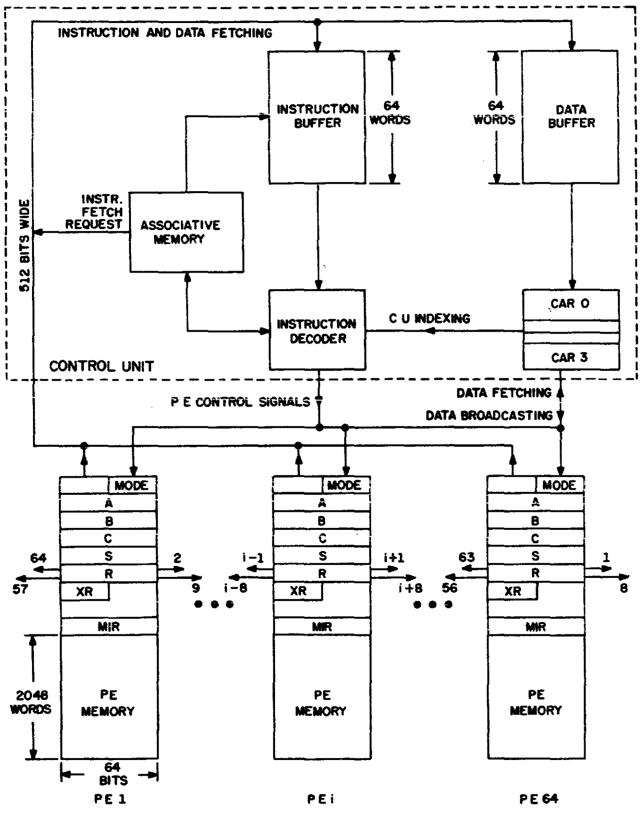


Figure 1—ILLIAC IV quadrant configuration into blocks for array storage

The TRANQUIL language has been designed to achieve both simpler specifications of, and explicit representation of the parallelism in, many algorithms, thus simplifying the programmer's task and maximizing the efficiency of computation on a computer such as ILLIAC IV. An overview of the software and application programming effort for the ILLIAC IV system has been given by Kuck.[2]

The TRANQUIL language

An important consideration in designing a language such as TRANQUIL is that the expression of parallelism in the language should be problem oriented rather than machine oriented. This does not, and should not, preclude programmer specification of data structure mapping at run time, but once the storage allocation has been made the programmer should have to think only in terms of the data structures themselves. Secondly, the means of specifying the parallelism should be such that all potential parallelism can be specified.

The structure of TRANQUIL is based on that of ALGOL; in fact, many ALGOL constructs are used with the addition of further data declarations, standard array operators, and revised loop specifications, including the addition of the set concept. Some of the ideas embodied in TRANQUIL follow similar constructs in other languages, e.g., the index sets in MADCAP[3] and the data structures and operators in APL.[4] The syntax of the current version of the TRANQUIL language is specified in Appendix B.

Data structures

The data structures which are recognized in TRANQUIL are simple variables, arrays and sets. All data structures, and quantities such as labels, switches and procedures, must be declared in some block head as in ALGOL. The data type attributes are *INTEGER*, *REAL*, *COMPLEX* and *BOOLEAN*. Certain precision attributes also may be specified.

A mapping function specification must be associated with every declaration of an array. The judicious choice of mapping functions is crucial to the efficient use of ILLIAC IV. Arrays must be mapped so as to optimize I/O transactions, minimize unfilled wasted areas of memory, and keep most of the PE's busy most of the time. In many array operations it is necessary to operate either on a whole row or a whole column of an array. All the PE's would be kept busy in the former case (one operand in each PE) but in the latter case all operands would normally be in only 1 PE. However, by specifying the skewed mapping function which rotates the i + 1st row across i PE's, columns as well as rows of the array can be accessed simultaneously. The more commonly used mapping functions such as *STRAIGHT*, *SKEWED*, *SKEWED PACKED*, and *CHECKER* are included in TRANQUIL. Array bounds may be specified dynamically, as in ALGOL, but all other attributes are nondynamic, for example:

REAL SKEWED ARRAY A[1:M, 1:N]

The user who wishes to specify his own mapping function may make use of a PE memory assignment statement. For example:

PEMEMORY PB [1:10, 1:64];

PEM FOR (I, J) *SIM* ([1, 2, . . ., 10] $\times$ [1, 2, . . ., 64]) *DO*

PB [I, J] $\leftarrow$ B [I, *MOD* (64, I + J − 1)];

REAL ARRAY (PB) B[1:10, 1:64];

where *SIM* is discussed in a later section, establishes virtual space of size 10 $\times$ 64 in PE memory, and then stores a 10 × 64 array B there in skewed form. Thus, instead of making up the aforementioned subarrays out of an array declaration, space reserved in PE memory may be used. In the program, the programmer refers to an element of memory space via the assigned array name B and its subscripts, as usual. It should be noted that storage mapping functions can not be specified dynamically. Should remapping of data be required, an explicit assignment statement may be used; e.g., to change the data in an array B from skewed to straight storage an assignment statement

$$A \leftarrow B$$

is used, where A has been declared to be a straight array.

A set is an ordered collection of elements each of which is an ordered n-tuple ($1 \leq n \leq 7$) and the set is said to be n-dimensional. A set declaration must specify one of the attributes *INCSET*, *MONOSET*, *GENSET*, or *PATSET* according as the set elements are to form an arithmetic progression (increment set), a strictly monotonic sequence (monotonic set), an arbitrary sequence (general set) or are multidimensional (pattern set), respectively, and in the latter case must also specify the size of n ($n > 1$). The declarations also may specify bounds for the integer values of the components of the n-tuple set elements, in ways analagous to the specification of array subscript bounds in ALGOL and FORTRAN, and an upper bound for the number of elements in the set. Some examples of set declarations are

INCSET	JJ
MONOSET	II [27, 6], KK [75, 75]
GENSET	A [150, 100]
PATSET	P(3) [0:20, 0:20, −5:15, 10]

The monotonic set II is to have at most 6 one-dimensional elements the integer values of which are to lie in the range [1,27]. The three-dimensional pattern set P is to have at most 10 3-tuple elements the first two components of which will lie in the range [0, 20] and the third components will be in [—5, 15].

Expressions

Expressions which can be formed in ALGOL can in general, also be formed in TRANQUIL. In addition arithmetic, logical and relational operators, and function designators may be used on arrays. The meaning of an operator is determined by its corresponding operands, which may be simple variables or arrays. All meaningful combinations of operands and attributes are valid; e.g., if A and B are matrices then A/B will mean $A \times B^{-1}$ if B has an inverse and the dimensions are correct. The result of a relational operator operating on two matrix operands is reduced to a single logical value through use of the qualifiers *ANY* and *ALL*, e.g.,

ANY A < B or *ALL* A < B

Examples of set definitional expressions are

SET1 ← [ ]

SET2 ← [1, 2, 3, 4, 5, 6, 7, 8]

SET3 ← [1, 2, . . ., 8]

SET4 ← [−2, P, Q, 25]

SET5 ← [[10, 10], [9, 8],[8, 6]]

where SET2 and SET3 are equivalent definitions and SET5 is a 2-dimensional pattern set. Replication factors may be used in general sets. For example:

SET6 ← [1(3), 4, 5(2)]

is equivalent to

SET6 ← [1, 1, 1, 4, 5, 5]

A useful device for the generation of a set is the run-time comparison of data values in parallel. For example, if A and B are vectors stored across PE memory,

$$A = \{-1, 3, 2, 10\} \text{ and } B = \{2, -3, 1, 12\},$$

then the operation

SET7 ← *SET* [I : A[I] < B[I]]

where I takes on the values 1, 2, 3, and 4 simultaneously, generates the set [1, 4], the order being defined as monotonically increasing. These definitions are readily extendible to multidimensional pattern sets which are generally used for picking scattered values in an array for simultaneous operation.

The set operators *INTERSECT*, *UNION*, and *COMPLEMENT* (a binary operation equivalent to the relative complement) may be used in TRANQUIL and always result in the generation of a monotonic set by reordering elements if necessary. The two additional set operators *CONCAT* and *DELETE* do not result in reordering of the elements. Some examples of set operations are:

if

R = [1, 2, 3, 4, 5]

S = [2, 4, 6, 8, 10]

T = [6, 4, 6, 5, 6, 7]

U = [100, 40, 0, 13]

then

R *UNION* U is [0, 1, 2, 3, 4, 5, 13, 40, 100]

R *CONCAT* S is [1, 2, 3, 4, 5, 2, 4, 6, 8, 10]

T *COMPLEMENT* R is [6, 7]

T *DELETE* R is [6, 6, 6, 7]

Finally, it is possible to create sets with multi-dimensional elements out of sets with scalar elements through use of the pair (,) and cartesian product ($\times$) set operators, e.g.,

[1, 2, 3, 4] , [2, 4, 6, 8] is [[1, 2,] [2, 4], [3, 6], [4, 8]]

[1, 2] $\times$ [3, 4] is [[1, 3], [1, 4], [2, 3], [2, 4]]

[1, 2] $\times$ [3, 4] , [5, 6] is [[1, 3, 5], [1, 4, 6], [2, 3, 5], [2, 4, 6]]

where , has higher precedence than $\times$.

Control statements

Control statements in TRANQUIL are used to designate sequential loops, simultaneous statement execution, and the usual conditional and unconditional transfers of control. Index sets play an integral part in these control statements. They are used as a source of values for iterative or loop control, and as a means of simultaneously specifying a number of array elements. Their association with the enablement and disablement of PE's should be obvious.

Sequential control: The SEQ statement

Sequential control refers to the existence of a loop through which designated index variables take on the values of a set or sets, one element at a time. It is written in the following general form:

FOR $(I_1, \ldots, I_n)$ *SEQ* $(II_1 \{\times \mid ,\} \ldots \{\times \mid ,\} II_n)$
$\{\langle\text{empty}\rangle \mid$ *WHILE* $\langle$Boolean expression$\rangle\}$ *DO* S

where { } means one of the alternatives separated by |, the scope is the statement S, n is an integer, $I_i (i = 1, \ldots, n)$ are control variables, and II_i $(i = 1, \ldots, n)$ are 1-dimensional set identifiers or literal set definitions. The use of this statement is illustrated by the following examples.

a. *FOR* (I, J) *SEQ* ([1, 2, . . ., 10], [5, 10, . . ., 50]) *DO*
   A[I] ← B[I + 1] + C[J]

is evaluated as

A[1] ← B[2] + C[5];

A[2] ← B[3] + C[10];

. . . . .

A[10] ← B[11] + C[50]

Note that the comma between the two set definitions denotes pairwise ordering for the control variables values.

b. *FOR* (I) *SEQ* ([2, 4, 6]) *WHILE* I < A[I] *DO*
   A[I] ← B[I] − A[I]

will continue looping until the Boolean expression is *FALSE* or the index set has been exhausted. As in ALGOL, no pass through the loop is made if the value of the Boolean expression is *FALSE* after the index variable is assigned the initial value of 2.

c. *FOR* (I, J) *SEQ* ([1, 2, 3, 4], [5, 6]) *DO*
   B[I, J] ← A[I, J]

In this case the difference in size of the two defined sets is resolved by considering only the pairs (1, 5) and (2, 6), that is, the exhaustion of the smallest index set signals the end of the loop. To indicate otherwise an asterisk is placed after the set the exhaustion of whose elements is to be used as the stopping condition. This means that any other sets which run out of elements before completion will be repeatedly used as many times as necessary. If the previous statement is rewritten as

FOR (I, J) *SEQ* ([1, 2, 3, 4]*, [5, 6]) *DO*
B[I, J] ← A[I, J]

the result is

B[1, 5] ← A[1, 5];

```
        B[2, 6] ← A[2, 6];
        B[3, 5] ← A[3, 5];
        B[4, 6] ← A[4, 6];

   d. FOR (I, J) SEQ ([1, 2] × [6, 7, 8]) DO
        A[I, J] ← B[J, I];
```

yields

```
        A[1, 6] ← B[6, 1];
        A[1, 7] ← B[7, 1];
        A[1, 8] ← B[8, 1];
        A[2, 6] ← B[6, 2];
        A[2, 7] ← B[7, 2];
        A[2, 8] ← B[8, 2];
```

where the lengths of the two sets do not create the problem that occurred with the pairwise operator. This example also illustrates that the frequency of element change is greatest for the rightmost set used.

Simultaneous control: The *SIM* Function and the *SIM* Statement

The parallel structure of ILLIAC IV is utilized in TRANQUIL by the specification of simultaneous control functions and statements. The general form of the *SIM* function is:

SIM BEGIN ⟨assignment statement⟩; . . .; ⟨assignment statement⟩ *END* where the enclosed assignment statements are executed simultaneously, i.e., the data used by any one of them are the data available before the *SIM* function was encountered.

The general form of the *SIM* statement for simultaneous control is:

FOR $(I_1, \ldots, I_n)$ *SIM* $(II_1 \{\times \mid ,\} \ldots \{\times \mid ,\}$ $II_n)$ *DO* S

where m, n are integers, I_i $(i = 1, \ldots, n)$ are control variables, II_i $(i = 1, \ldots, m)$ are k-dimensional sets $(0 < k \leq 7)$, n equals the total number of dimensions of all II_i, and S is a statement. For this statement each substatement S_i of S is executed with the data available before it is reached, i.e., just as if a *SIM* function was placed around each S_i. In this regard it is important to note that simultaneous control is not loop control, but designates that each S_i is to be executed in parallel and thus the order of the associated sets is not important.

Some examples of the use of *SIM* are:

```
   a. FOR (I, J) SIM ([1, 2, 3]*, [4, 5]) DO
        A[I, J] ← B[J, I]
```

is evaluated as

```
      SIM BEGIN A[1, 4] ← B[4, 1];
                A[2, 5] ← B[5, 2];
                A[3, 4] ← B[4, 3]
         END

   b. FOR (I, J) SIM (II, × JJ) DO
        BEGIN
          C[I, J] ← 0;
          FOR (K) SEQ (KK) DO
          C[I, J] ← C[I, J] + A[I, K] × B[K, J]
        END
```

is a general routine for the multiplication of two com patible matrices A and B if the index sets II and JJ specify the rows of A and the columns of B, respectively

It should be noted that when a set is used in a sequential or simultaneous control statement, it cannot be altered within the scope of that statement.

Nested *SEQ* and *SIM* statements

The *SEQ* and *SIM* control statements described above may be nested. The effect of nesting is clear except when a *SIM* statement occurs within the scope of another *SIM* statement, in which case statements inside the scope of both are executed under the control of sets which are related by the cross product operator, for example:

```
   FOR (I, J) SIM (II, JJ) DO
      BEGIN FOR (K) SIM (KK) DO
            BEGIN
              Area A
            END;
          . . .
      END
```

where the control statement in effect in area A is, in effect,

FOR (I, J, K) *SIM* (II, JJ × KK) *DO*

If clauses

General forms:

a. *IFSET* ⟨indexed Boolean expression⟩ *THEN*
b. *IF* {*FOR* ($I_1, \ldots, I_n$) *SIM* (II_1 {×|,} . . . {×|,} II_m)| ⟨empty⟩} {*ANY*|*ALL*| ⟨empty⟩} ⟨Boolean expression⟩ *THEN*

If clauses may be used in arithmetic, Boolean, set and designational expressions. The Boolean expression in form (a) must involve a control variable under *SIM* control and thus not have a single logical value. This is meaningful in arithmetic and Boolean expressions of assignment statements having left parts which use the same control variable, and also in conditional statements where the control variable is used in the left part of some associated assignment statement. An example of this use is:

```
FOR  (I) SIM ([1, 2, . . ., 100]) DO

        T[I] ← IFSET A[I] < B[I] THEN A[I]
        ELSE B[I]
```

is equivalent to

```
FOR  (I) SIM ([1, 2, . . ., 100]) DO

        IFSET A[I] < B[I] THEN T[I] ← A[I]

                          ELSE T[I] ← B[I]
```

In either form T[I] ← A[I] for all values of I for which the value of the Boolean expression is *TRUE* and T[I] ← B[I] otherwise.

The form (b) results in only a single Boolean value based on the *ANY* or *ALL* modifier, and the scope of the *SIM* control (if explicitly present) extends over the Boolean expression only. If the vector A of length 2 has elements 5 and 10, the if clause test

IF FOR (I) *SIM* ([1, 2]) *ANY* A[I] < 7

has the value true since A[1] < 7. The same result is achieved by use of the if clause

IF ANY A < 7

The TRANQUIL compiler and its implementation

Introduction

The syntax of TRANQUIL has been specified in a form which is accepted by the syntax preprocessor of the Translator Writing System (TWS)[5,6,7,8] being built at the University of Illinois. The preprocessor automatically generates the parsing algorithm for the compiler. In pass 1 of the compiler the recognition of source code constructs invokes calls, via the action numbers embedded in the syntax definition, to semantic actions. These actions build descriptor tables containing, in part, information about declaration types, attributes, and block structure, and transform the source code into an intermediate language form which is composed of operators and operands, the latter being references to the descriptor tables.

Pass 2 is the main body of the compiler. The intermediate language stream is read and operators call pass 2 semantic actions (on the basis of their context) for generation of assembly language instructions using associated operands. A number of other important considerations arise, among which are the storage of arrays and sets, and the efficient allocation and use of CU registers. These problems and some solutions are discussed in the following sections.

Array blocking and storage allocation

The compiler partitions all arrays into 2-dimensional blocks the maximum size of which is 64q × 64q words (q = 1, 2, or 4 according as 1, 2 or 4 quadrants of 64 PE's are being used) since ILLIAC IV may be regarded as an array with 2048 rows (number of words in a PE) × 64q columns (number of PE's). For the purposes of this section it will be assumed that q = 1. The sizes of the blocks obtained by the array partitioning then fall into 4 categories:

a. 64 × 64
b. m × 64
c. 64 × n
d. m × n   m, n < 64

which are called SQUARE, HBLOCK, VBLOCK and SBLOCK, respectively. Small blocks belonging to the same array are packed together to form a larger block, details of which are given by Muraoka.[9] Figure 2 illustrates the partitioning of a 3-dimensional array into 12 blocks and Figure 3 illustrates how the smaller subblocks are packed together to form larger blocks. The partitioning of an array into blocks is independent of the mapping function; i.e., for a *SKEWED* array skewing is done after partitioning.

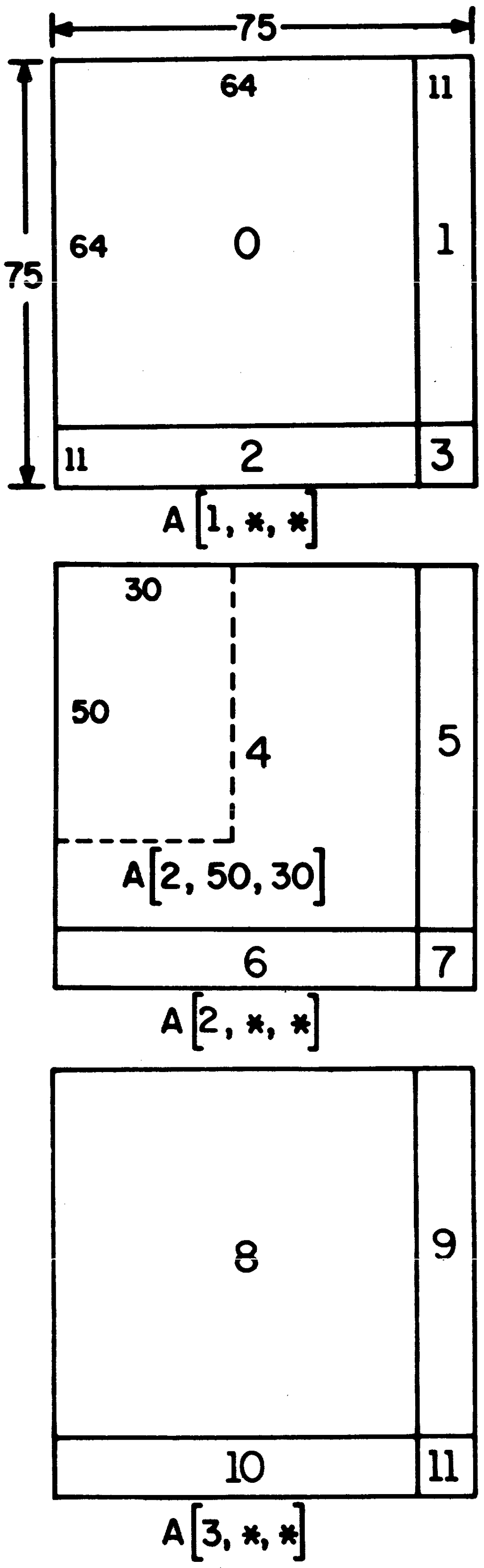


Figure 2—The partitioning of array A[1:3, 1:75, 1:75] into blocks for array storage

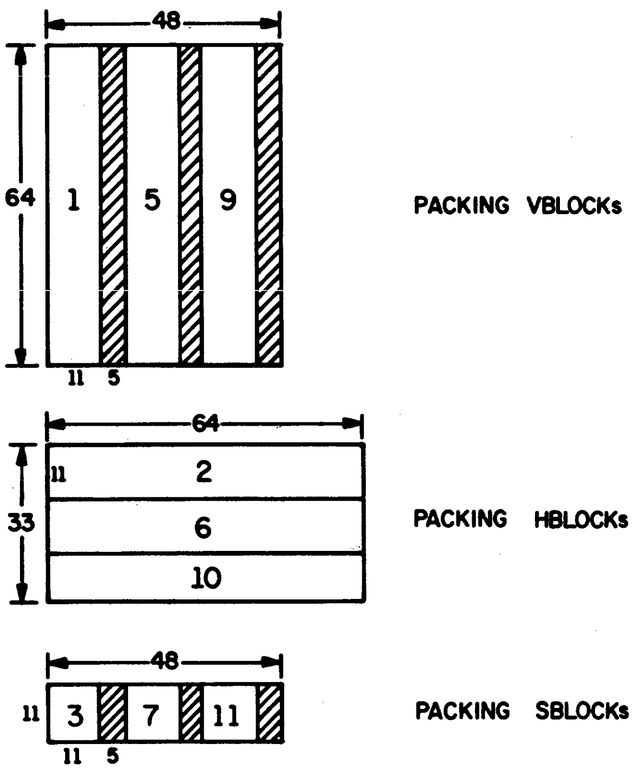


Figure 3—Block packing for array A[1:3, 1:75, 1:75]

All array operations and data transfers between ILLIAC IV disk and PE memory are done in terms of these blocks. A block of size m × n may be placed in any m adjacent words (rows) stored in n adjacent PE's in PE memory. Blocking facilitates the use of arrays which are larger than the PE memory. All data are normally stored on the 10^9 bit ILLIAC IV disk and blocks are only brought into PE memory (at a transfer rate of .5 × 10^9 bits/second) as required. The TRANQUIL compiler automatically generates block transfer I/O requests, which are handled by the operating system. Thus, it is possible to write a TRANQUIL program which includes no explicit data transfers.

The effective address of any array element is obtained by computing the block number, which determines an absolute base address (the address of the upper leftmost element of the block if the block is in memory) and a relative address within that block. The block number for an element $A[i_1, i_2, \ldots, i_{n-1}, i_n]$ of an array declared

$A[\ell_1 : u_1, \ldots, \ell_n : u_n]$ is given by

$$((\ldots((i_1 - \ell_1)M_2 + (i_2 - \ell_2))M_3$$
$$+ \ldots (i_{n-2} - \ell_{n-2}))M'_{n-1} + i'_{n-1})M'_n + i''_n ,$$

where

$$M_i' = u_i - \ell_i + 1,$$

$$M_i'' = (M_i' + 63)\,div\,64,\; i_k'' = (i_k - \ell_k)\,div\,64\;.$$

If the array is *SKEWED* the relative PE number and relative PE address of the element in the specified block are given by

$$[(i_{n-1} + i_n) - (\ell_{n-1} + \ell_n)] \bmod 64$$

and

$$(i_{n-1} - \ell_{n-1}) \bmod 64\;,$$

respectively. After skewing of the blocks in Figure 2, the element A[2, 50, 30] is specified by the block number, and PE number and PE address relative to the base address of the block, which have values 4, 14 and 49, respectively. In most cases some or all of the elements in a row or column are used simultaneously. In the case of column operation, for example, each PE can simultaneously compute the relative address (index value) which it will require.

In allocating memory space for a block a linked space list, which keeps track only of the number of rows of memory which have been used, is utilized. If a block of size m $\times$ 64 is to be stored, the list is searched to locate a space of m adjacent rows and the block is stored there. In the case of a block of size m $\times$ n (m, n $<$ 64) 64 rows of PE memory may be allocated and a sublist corresponding to this 64 $\times$ 64 block of storage is established. The sublist consists of a Boolean array in which each bit represents the use or otherwise of each 8 $\times$ 8 subblock of the associated 64 $\times$ 64 block of storage, thus allowing several small blocks to be stored together in a larger unit.

The storage and use of sets

Associated with the introduction of sets in TRANQUIL is the task of finding storage schemes which can be used efficiently. Sets can be used for loop control, for enabling or disabling PE's, or for PE indexing. The increment set can be used in all three ways and is stored using two words per set. One word contains the first element, the increment, and the limit packed for use by CAR instructions. The other word is used as a bias value in the case where negative elements are, or may be, in the set.

When an increment set is used for sequential control, CU test and increment instructions operate on the appropriate CAR register. When an increment set is used for simultaneous control, it can be expanded at run time into mode words or explicit numbers. Mode words are 64-bit words used to store the elements of a set by position number, where a 1 in the n-th bit indicates that n is an element of the set. These words are most frequently used in PE mode registers to enable and disable PE's. Mode words can be generated from an increment set by using a memory row that contains the PE numbers in ascending and descending order and regular mode patterns similar to the 4 PE system shown in Figure 4. In the figure the mode pattern was formed by considering the b_0 bits to be all ones, the b_1 bits to be alternating zeros and ones, the b_2 bits to be two zeros alternating with a one, and finally the b_3 bits to be three zeros alternating with a one. In the general 64 PE case, the word in the i-th PE is

Mode pattern	i	63-i

where the 32-bit mode pattern results by considering the b_0 bits to be all ones: i.e., all PE's on when in use, the b_1 bits having the pattern 0101. . ., the b_2 bits 001001. . ., and the b_i bits 0_i10_i1 where 0_i stands for i zeros.

Now consider the example of expanding the increment set JJ of Appendix A into mode words and explicit numbers. The set JJ is used simultaneously in forming the comparison set KK by a Boolean test on the skewed arrays A and B. For a given I every other element of A, as signified by JJ, is moved to the A register in the PE's, using the base address of A that has been brought to the CU data buffer as indicated in Figure 5. Every other element corresponds to the b_1 bit mode pattern, this pattern appearing in the PE mode positions of Figure 5 which is based on Figure 1. The case for I = 2 is shown. Every other element of the I-th column of B is moved to the B register. In this case every other ascending PE number is used in the PE index registers, XR, with appropriate routing to account for the skewed storage. For I = 2

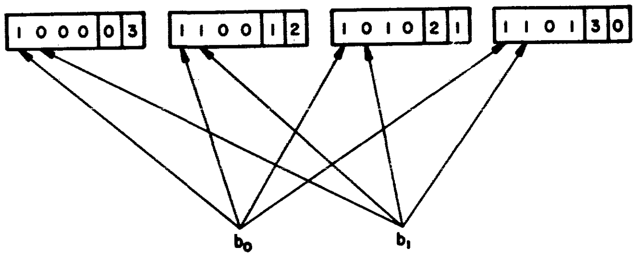


Figure 4—Mode patterns and explicit values for increment sets

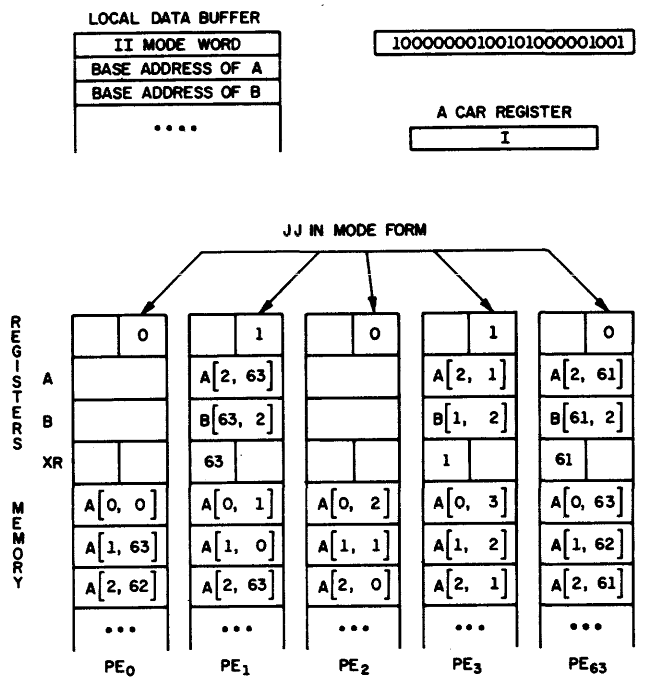


Figure 5—PE and CU status for I = 2 in the example problem

in Figure 5, an end around route of two is necessary. Every other element of a column is fetched to the B register again by the use of JJ in mode form.

The sets II and KK in Appendix A are examples of monotonic sets and are stored in mode form. For looping on II the CU does leading ones detection on the II mode pattern, illustrated in Figure 5, to determine the explicit set elements used in the CAR as CU indexes for array A, and KK is used in mode form in the PE mode registers under simultaneous control. Monotonic sets can also be stored as explicit numbers for index use. The general set is always stored in explicit form, for obvious reasons, and pattern sets are stored as mode words.

The actual management of mode and explicit storage schemes involves the use of a permanent storage area and a stack. All set definitions are stored in the permanent storage area except comparison sets. The stack is used for storing current set values that are obtained from the permanent area or generated by the user or compiler. The stack is also used because program flow is not known at compile time, in general, due to data based branches, and thus changes in these sets are unpredictable. In PE memory the storage of either the mode or explicit set representation begins on an 8 word boundary to make best use of the 8 word CU fetch capabilities. Further details of the handling of sets are given by Wilhelmson.[10]

CU storage allocation and use

The allocation and use of CU registers is a very important ILLIAC IV problem since CU instructions which cannot be overlapped with PE instructions leave all PE's idle. The allocation of CU registers for needs known at compile time is accomplished by calling one of a group of procedures that have an underlying allocation priority system and that use compile-time pointers to and from important tables and storage locations. The local data buffer is divided into two parts: the lower part for use as determined at compile time and the upper part for dynamic use at run time. The lower part is further divided into three parts. The first is one word used for bit storage and later testing. A list of free bits is kept, bits being assigned on a space available basis. The second part is 16 words in length, where the use of this space is kept to low priority requests unless space is needed for high priority requests. Priorities range from 0 to 3 and are assigned by the compiler writer, these assignments being based on intuition and experience. The third part has n words ($0 \leq n \leq 47$), where the optimal value of n is yet to be determined. The space is used as much as possible for high priority requests. The reason for this device is to try and keep low priority requests in the lower area, since the lower area will be used to store 8-word blocks transmitted to the CU. When the local data buffer becomes full a word is freed, with appropriate storage and pointer modification if necessary. Thus a user can let the procedures free words when necessary unless he wishes to do so earlier.

Three of the CAR registers are allocated using the same priorities as in the local data buffer. The fourth is a free register which may be used in a variety of ways.

Another CU problem is connected with its particular data composition, and results from transfers. For example, at the beginning of a loop data in the CU has a certain composition. This composition should be reinstated each time through the loop. This is made possible by remembering the composition of the CU data at the beginning of the loop.

For backward transfers code to set up the CU properly is placed at the jump point while for forward transfer code is placed at the location transferred to. The priority scheme may appear to add to the problem of moving words in and out of CU memory at transfer points. This scheme has been developed since 8-word block stores to PE memory are not allowed; only one word at a time can be stored in PE memory by the CU.

Assignment statements

The use of sets, the notion of *SIM*, the number of different types of arithmetic and storage schemes, combined with the need to compile efficient code for a parallel machine necessitate a substantial analysis of each assignment statement. We now consider this analysis as it is carried out in pass 2 of the compiler.

The analysis is effected in several passes over the postfix intermediate language. Consider the last assignment statement in the example in Appendix A:

$$A[I, K] \leftarrow A[I + 1, K] + B[I, K + 1] ;$$

Before we even begin to generate code a decision must be made as to which index is to be processed simultaneously (i.e., across the PE's) and which is to be done sequentially. The first pass over the intermediate language determines this and also copies the intermediate language into a table to be used for future passes. When a set linked* identifier is entered in the table, additional information provided by the set definition or declaration is also entered. In the case of I, which is linked via *SIM* control to II, the set is known exactly and precise information from the set definition is entered in the table. For K the compiler makes an estimate of the size and density of the set based on the upper bounds given in the declaration of KK.

In general, when operations are performed on pairs of subscripts or pairs of subscripted arrays, information about the interaction between these subscripts must be generated. For example, in the case of the subscript expression I + 1 in the example above, the addition of 1 in no way alters the size, density, or type of the set. Thus, the information provided for I will be recopied with the + operand.

After the subscript expression has been processed, a check is made to see how well the type of set resulting from the index expression will work with the particular dimension of the array involved. In the example there are only two-dimensional skewed arrays in which either columns or rows can be easily accessed in parallel. If one of the arrays were straight, then at this point it would be discovered that no set will work well for the column index, because each column is stored in a single PE. This information plus information about the set density, set size, and the array size are all combined to compute a probable efficiency; i.e., the number of PE's that will probably be enabled if this index were varied simultaneously. Of course, it is easy to think up cases in which the estimate will be totally wrong, but in most practical cases encountered, the estimate is reasonable. A table of these probable efficiencies is generated for each set. If the set appears in different subscripts, then on the second occurrence the new estimate is set to the minimum of the previous and present estimates.

* We say that I is set linked to II in a statement like *FOR* (I) *SIM* (II) *DO*.

When the end of the assignment statement is reached, the table of probable efficiencies is sorted and the result of this determines the order in which the indexes will vary. In the example K will be the index chosen to vary across the PE's because the set II is known to be small (6 elements) and the declaration of KK holds the probability of it being fairly large. Now an outer loop must be compiled to generate sequentially the elements of II. Finally, the remainder of the statement is compiled. The effect of the code that is compiled for the example assignment statement follows.

One local data buffer location is set aside as an index to the mode words of KK. Four more locations are set aside for the base addresses of the subblocks of the arrays A and B. The first mode word for KK is loaded and the leading ones detector is used to set the first value of K. This value, plus the base address of A, plus 1, plus the index set 0, 1, 2, . . ., 63 in the PE index registers is used to access the first column of A. In a similar manner the address for the first row of B is fetched, loaded into RGR and a route left one PE is performed The addition is executed and the first mode word for KK is used to store the result in A. Now the same process is repeated for the next subblock of A, except that the mode pattern for KK must be ended with a word having 11 1's followed by 0's, because the second subblock of A is only 11 words wide. Additional complications, such as pairwise *SIM* control specification, small subarrays, and *SIM* blocks add to the complexity, but not to the substance, of the algorithm outlined above.

It is clearly impossible to efficiently compile a single short assignment statement for ILLIAC IV, but it is conceivable that a large number of simple assignment statements could be integrated into a fairly efficient ILLIAC IV program. Incorporating such a feature into a compiler presents two basic problems. The first is an algorithm for efficiently integrating a large number of interrelated assignment statements. Ordinarily the simple assignment statements will be scattered throughout the program. Also, many of the sequential calculations that are prime targets for an integration scheme are likely to be embedded as subexpressions in assignment statements containing *SIM* controlled variables. Filtering out and gathering together these candidates

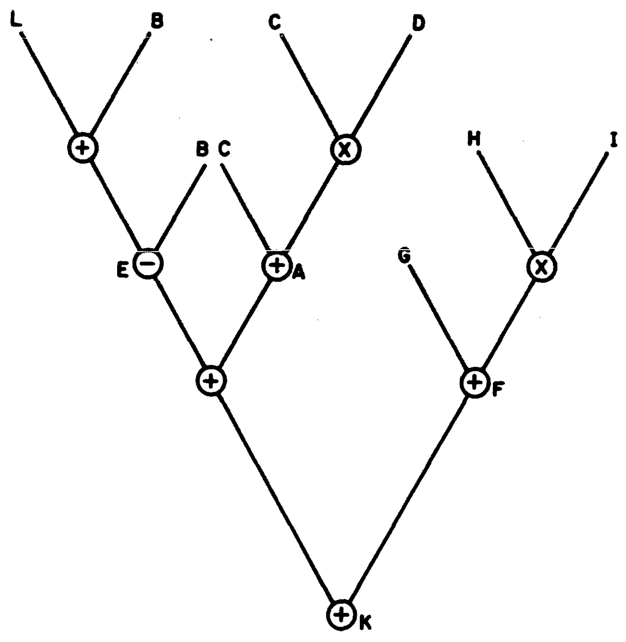


Figure 6—The tree structure for a set of interrelated arithmetic statements

for the integration scheme constitutes the second problem.

Figure 6 is a tree structure for the set of assignment statements:

$$A \leftarrow B + C \times D$$

$$E \leftarrow L + B - C$$

$$F \leftarrow G + H \times I$$

$$K \leftarrow A + E + F$$

No node on this tree can be calculated until all nodes on subbranches have been calculated. The method of computing such a tree on ILLIAC IV involves first mapping assignment statements into PE's, in a more or less arbitrary manner. The assignment statements are restricted to a small number of operations like addition, subtraction, multiplication and division. ILLIAC IV can only perform one of these operations at a time. A count of the number of PE's that can take advantage of each of these operations is made and that operation which will be executed by the most PE's is the one that code is compiled for. Then the PE counts for all operations are revised and the process continues until all calculations have been performed. A similar algorithm is used to do routing to bring the results computed in one PE to the PE's where they are needed. This algorithm is invoked whenever the number of PE's eligible for any operation falls below a certain limit.

The problem of gathering together assignment statements for processing by this method is many faceted. What is desired is a rearrangement of the program where simple assignment statements, simple subexpressions, and simple expressions generated by the compiler, like address calculation, have been brought together at several collection points. To rearrange code in this manner requires an extensive analysis of the overall program to determine what subexpressions and statements can be moved, and how far.

This analysis is carried out at the intermediate language level. The collection points are determined to be at the beginning of blocks, subexpressions are moved as physically high up in the code as possible, except that they are not moved past a block head unless they can be moved to the head of an outer block. The method produces a number of bonuses. Calculations inside loops tend to be moved outside when logically permissible. Thus, it is profitable to move nonsimple subexpressions also. Further, duplicate subexpressions can easily be eliminated because they tend to gather at the same point. Finally, for each block a record is made of what variables are nondynamic within that block. Thus, in pass 2, any expressions generated using these variables can be added to the collection of subexpressions at the beginning of the appropriate block. At the head of this block, a transfer to the end of the block is compiled, and when all code in the body of the block has been generated, the complete collection of assignment statements is compiled followed by a transfer back to the beginning of the block. More details of this type of analysis are given by Budnik.[11]

SUMMARY

Designing and implementing a language and its compiler for ILLIAC IV presents a number of problems not encountered with procedure oriented languages for sequential machines. In the design of the language these problems have been met, primarily through the use of sets as indexes and the introduction of language elements for explicit denotation of simultaneous operations. Experience has shown that the resulting notation is as easy to learn as that of conventional languages and in most instances it is more concise.

The task of efficiently compiling a language such as TRANQUIL for the ILLIAC IV is more difficult than compiling for conventional machines, simply because the standard compiling techniques are inadequate, thus requiring new compilation algorithms to be invented. These techniques will undoubtedly be refined as further experience is gained with the use of ILLIAC IV and parallel languages. However, the completion of the major parts of the TRANQUIL compiler has already demonstrated that reasonably efficient object code can be generated for a large class of array-type problems which have been programmed in TRANQUIL.

Several features of TRANQUIL have been omitted from this paper, notably input/output statements and procedures. Execution time input/output will be from/to the ILLIAC IV disk (secondary storage) in blocks of data. Most of these data transfers will be implicitly specified in TRANQUIL programs. However, some explicit specification of unformatted data transfers will be provided. The provision of additional software to facilitate format specified transfer of data between external peripherals (tertiary storage) and the ILLIAC IV disk is planned. The specification of procedure declarations, and their use in a parallel environment, is under investigation. Additional features being considered for later incorporation into the TRANQUIL compiler include overlayable code segments, quadrant configuration independent code, and more specialized data structures and mapping functions. Although aspects of the language and its compiler are still being developed, it has been demonstrated that TRANQUIL is a highly satisfactory and useful complement to the ILLIAC IV hardware system design.

ACKNOWLEDGMENTS

The research reported in this paper was supported in part by the Department of Computer Science, University of Illinois at Urbana-Champaign, and in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center, under Contract No. USAF 30(602)4144.

REFERENCES

1 G BARNES R BROWN M KATO D KUCK D SLOTNICK R STOKES
The ILLIAC IV computer
IEEE Transactions on computers Vol C-17 746-757 August 1968

2 D J KUCK
ILLIAC IV software and applications programming
IEEE Transactions on computers Vol C-17 August 1968 758-770

3 M B WELLS
Aspects of language design for combinatorial computing
IEEE Transactions on computers Vol EC-13 431-438 August 1964

4 K E IVERSON
A programming language
John Wiley New York 1962

5 R S NORTHCOTE
The structure and use of a compiler-compiler system
Proc 3rd Australian computer conference 1966

6 F L DEREMER
On the generation of parsers for BNF grammars: an algorithm
Proc S J C C 1969

7 A J BEALS
The generation of a deterministic parsing algorithm
Report No 304 Dept of Computer Science
University of Illinois at Urbana 1969

8 H R G TROUT
A BNF like language for the description of syntax directed compilers
Report No 300 Dept of Computer Science
University of Illinois at Urbana 1969

9 Y MURAOKA
Storage allocation algorithms in the TRANQUIL compiler
Report No 297 Dept of Computer Science
University of Illinois at Urbana 1969

10 R B WILHELMSON
Control statement syntax and semantics of a language for parallel processors
Report No 298 Department of Computer Science
University of Illinois at Urbana 1969

11 P P BUDNIK
TRANQUIL arithmetic
Report No 296 Dept of Computer Science
University of Illinois at Urbana 1969

APPENDIX A: A SAMPLE TRANQUIL PROGRAM

```
BEGIN
REAL SKEWED ARRAY A, B[1:75, 1:75];
INCSET JJ;
MONOSET II(1) [27, 6], KK(1) [75, 75];
INTEGER I, J, K;
II ← [2, 10, 13, 15, 21, 24];
JJ ← [2, 4, . . ., 74];
FOR (I) SEQ (II) DO
        BEGIN FOR (J) SIM (JJ) DO
                KK ← SET (J: A[I, J] < B[J, I]);
                FOR (K) SIM (KK) DO
                    A[I, K] ← A[I + 1, K] + B[I, K + 1]
        END;
FOR (I, K)  SIM (II × KK) DO
                A[I, K] ← A[I + 1, K] + B[I, K + 1]
END
```

APPENDIX B: A SPECIFICATION OF THE SYNTAX OF TRANQUIL

A brief description of the syntax metalanguage is given in APPENDIX C.

B.1 *Program*

```
⟨PROGRAM⟩ ::= ⟨BLOCK⟩
⟨BLOCK⟩ ::= BEGIN list [⟨DECLARATION⟩ ⋕;]
                 list ⟨STATEMENT⟩ separator ⋕;
                 [⋕;] END;
⟨STATEMENT⟩ ::= ⟨NONEMPTY STATEMENT⟩ | ⟨ ⟩;
⟨NONEMPTY STATEMENT⟩ ::= [⟨ *I⟩ : ]*
                 [⟨CONTROL STATEMENT⟩ |
                 GO TO ⟨DESIGNATIONAL EXPRESSION⟩ |
                 BEGIN ⟨NONEMPTY STATEMENT⟩ [⋕; ⟨STATEMENT⟩]* END |
                 ⟨BLOCK⟩ |
                 ⟨IF CLAUSE⟩ ⟨STATEMENT⟩ [ELSE ⟨STATEMENT⟩]& |
                 ⟨ASSIGNMENT STATEMENT⟩];
```

B.2 *Declarations*

```
⟨DECLARATION⟩ ::= ⟨VARIABLE DECLARATION⟩ |
                 ⟨ARRAY DECLARATION⟩ |
                 ⟨PEM RESERVE DECLARATION⟩ |
                 ⟨PEM ASSIGNMENT DECLARATION⟩ |
                 ⟨SET DECLARATION⟩ |
                 ⟨SWITCH DECLARATION⟩ |
                 ⟨LABEL DECLARATION⟩;
```

B.2.1 *Variable Declarations*

```
⟨VARIABLE DECLARATION⟩ ::= ⟨ATTRIBUTE⟩
                 [COMPLEX]& list ⟨*I⟩
                 separator, ;
⟨ATTRIBUTE⟩ ::= BOOLEAN | REAL | REALS | REALD | INTEGER |
                 INTEGERS | INTEGERL | BYTE8 | BYTE16 ;
```

B.2.2 *Array Declarations*

```
⟨ARRAY DECLARATION⟩ ::= [⟨ATTRIBUTE⟩]& [⟨MAPPING FUNCTION⟩]&
                 ARRAY ⟨ARRAY LIST⟩ | [⟨ATTRIBUTE⟩]& ARRAY
                 (⟨PEM AREA⟩) ⟨ARRAY LIST⟩;
⟨MAPPING FUNCTION⟩ ::= STRAIGHT | SKEWED | SKEWED PACKED | CHECKER;
⟨ARRAY LIST⟩ ::= list [list ⟨*I⟩ separator, ⟨BOUND LIST⟩]
                 separator,;
⟨BOUND LIST⟩ ::= ⋕[[list [[ * | ** | ⋕ | ⋕⋕ ]&
                 ⟨ARITHMETIC EXPRESSION⟩ : ⟨ARITHMETIC EXPRESSION⟩]
                 separator, | list [[ * | ** | ⋕ | ⋕⋕ ]&
                 ⟨ARITHMETIC EXPRESSION⟩] separator,] ⋕];
⟨PEM AREA⟩ ::= ⟨*I⟩;
```

```
B.2.3 PEM Reserve Declarations
      ⟨PEM RESERVE DECLARATION⟩ ::= PEMEMORY ⟨PEM AREA NAME⟩
                          ⋇[⟨UNSIGNED INTEGER⟩,⟨UNSIGNED INTEGER⟩⋇];
      ⟨PEM AREA NAME⟩ ::= ⟨*I⟩;
      ⟨UNSIGNED INTEGER⟩ ::= ⟨* N⟩;

B.2.4 PBM Assignment Declarations
      ⟨PEM ASSIGNMENT DECLARATION⟩ ::=
                          PEM [⟨PEM ASSIGNMENT CONSTRUCT⟩ |
                          BEGIN list ⟨PEM ASSIGNMENT CONSTRUCT⟩
                          separator ; END];
      ⟨PEM ASSIGNMENT CONSTRUCT⟩ ::=
                          ⟨PEM ASSIGNMENT STATEMENT⟩ |
                          ⟨SET ASSIGNMENT STATEMENT⟩ | ⟨PEM FOR STATEMENT⟩;
      ⟨PEM ASSIGNMENT STATEMENT⟩ ::= ⟨* I⟩
                          ⋇[[⟨UNSIGNED INTEGER⟩ | ⟨* I⟩],
                          [⟨UNSIGNED INTEGER⟩ | ⟨* I⟩] ⋇] ← ⟨* I⟩
                          ⋇[list ⟨ARITHMETIC EXPRESSION⟩ separator, ⋇];
      ⟨PEM FOR STATEMENT⟩ ::=
                          FOR (⟨SET VARIABLE LIST⟩) SIM
                          (⟨SET NAME LIST⟩) DO
                          [⟨PEM ASSIGNMENT CONSTRUCT⟩ |
                          BEGIN list ⟨PEM ASSIGNMENT CONSTRUCT⟩
                          separator; END];

B.2.5 Set Declarations
      ⟨SET DECLARATION⟩ ::= [INCSET | MONOSET | GENSET | PATSET] list
                          ⟨SET SEGMENT⟩ separator,;
      ⟨SET SEGMENT⟩ ::= list ⟨* I⟩ separator,
                          [(⟨* N⟩)]& [⋇[list [⟨ARITHMETIC EXPRESSION⟩
                          [: ⟨ARITHMETIC EXPRESSION⟩ ]&]
                          separator, ⋇] ]&;

B.2.6 Label and Switch Declarations
      ⟨LABEL DECLARATION⟩ ::= LABEL list ⟨*I⟩ separator,;
      ⟨SWITCH DECLARATION⟩ ::= SWITCH ⟨*I⟩ ← list
                          ⟨DESIGNATIONAL EXPRESSION⟩ separator,;

B.3.1 Control Statements
      ⟨CONTROL STATEMENT⟩ ::= FOR (⟨SET VARIABLE LIST⟩)
                          [SEQ | SIM] (⟨SET NAME LIST⟩) DO ⟨STATEMENT⟩ |
                          FOR (⟨SET VARIABLE LIST⟩) SEQ (⟨SET NAME LIST⟩)
                          WHILE ⟨BOOLEAN EXPRESSION⟩ DO
                          ⟨STATEMENT⟩ | ⟨SIM BLOCK⟩;
      ⟨SET VARIABLE LIST⟩ ::= list ⟨*I⟩ separator,;
      ⟨SET NAME LIST⟩ ::= list [⟨*I⟩ [⋇ *]&
                          | ⟨SET DEFINITION TAIL⟩] separator [, | ×];
      ⟨SIM BLOCK⟩ ::= SIM BEGIN list ⟨ASSIGNMENT STATEMENT⟩
                          separator ⋇ ; [⋇ ;]& END;

B.3.2 Set Definitions
      ⟨SET DEFINITION TAIL⟩ ::= ⋇[⟨LIST SET⟩ ⋇] |
                          ⟨COMPARISON SET⟩;
      ⟨LIST SET⟩ ::= ⟨ELEMENT⟩ [,⟨ELEMENT⟩,. . . , ⟨ELEMENT⟩ |
                          , ⟨ELEMENT⟩ ]* | ⟨⟩;
      ⟨ELEMENT⟩ ::= ⋇[list ⟨ARITHMETIC EXPRESSION⟩ separator, ⋇] |
```

⟨ARITHMETIC EXPRESSION⟩
[(⟨ARITHMETIC EXPRESSION⟩)]&;
⟨COMPARISON SET⟩ ::= *SET* ⋕[*list* ⟨*I⟩ *separator*,:
⟨BOOLEAN EXPRESSION⟩ ⋕];

B.4 *Designational Expressions*
⟨DESIGNATIONAL EXPRESSION⟩ ::= ⟨SIMPLE DESIGNATIONAL EXPRESSION⟩ |
⟨IF CLAUSE⟩ ⟨SIMPLE DESIGNATIONAL EXPRESSION⟩
ELSE ⟨DESIGNATIONAL EXPRESSION⟩;
⟨SIMPLE DESIGNATIONAL EXPRESSION⟩::= (⟨DESIGNATIONAL EXPRESSION⟩) |
⟨SWITCH IDENTIFIER⟩ ⋕[⟨ARITHMETIC EXPRESSION⟩ ⋕] |
⟨LABEL IDENTIFIER⟩;
⟨SWITCH IDENTIFIER⟩ ::= ⟨*I⟩;
⟨LABEL IDENTIFIER⟩ ::= ⟨*I⟩;
IF Clauses
⟨IF CLAUSE⟩ ::= *IF* [⟨CONTROL HEAD⟩]& [*ANY* | *ALL*]&
⟨BOOLEAN EXPRESSION⟩ *THEN* |
IFSET ⟨BOOLEAN EXPRESSION⟩ *THEN*;
⟨CONTROL HEAD⟩ ::= *FOR* (⟨SET VARIABLE LIST⟩) *SIM* (⟨SET NAME LIST⟩);

B.6 *Assignment Statements*
⟨ASSIGNMENT STATEMENT⟩ ::=
⟨BOOLEAN ASSIGNMENT STATEMENT⟩ |
⟨ARITHMETIC ASSIGNMENT STATEMENT⟩ |
⟨SET ASSIGNMENT STATEMENT⟩;

B.6.1 *Boolean Assignment Statements*
⟨BOOLEAN ASSIGNMENT STATEMENT⟩ ::= *list* [⟨*I⟩ ←]
⟨BOOLEAN EXPRESSION⟩;
⟨BOOLEAN EXPRESSION⟩ ::= ⟨SIMPLE BOOLEAN⟩ | ⟨IF CLAUSE⟩
⟨SIMPLE BOOLEAN⟩ *ELSE* ⟨SIMPLE BOOLEAN⟩;
| SIMPLE BOOLEAN⟩ ::= ⟨BOOLEAN FACTOR⟩ [[*OR* | *IMP* | *EQV*]
⟨BOOLEAN FACTOR⟩]*;
⟨BOOLEAN FACTOR⟩ ::= ⟨BOOLEAN SECONDARY⟩
[*AND* ⟨BOOLEAN SECONDARY⟩]*;
⟨BOOLEAN SECONDARY⟩ ::= ⟨BOOLEAN PRIMARY⟩ | *NOT* ⟨BOOLEAN PRIMARY⟩;
⟨BOOLEAN PRIMARY⟩ ::= *TRUE* | *FALSE* | ⟨*I⟩ | ⟨RELATION⟩ |
(⟨BOOLEAN EXPRESSION⟩);
⟨RELATION⟩ ::= ⟨ARITHMETIC EXPRESSION⟩ *ELT* ⟨SET EXPRESSION⟩ |
⟨ARITHMETIC EXPRESSION⟩ ⟨RELATIONAL OPERATOR⟩
⟨ARITHMETIC EXPRESSION⟩ | ⟨SET EXPRESSION⟩
[= | *EQL* | ≠ | *NEQ*] ⟨SET EXPRESSION⟩;
⟨RELATIONAL OPERATOR⟩ ::= *LSS* | *LEQ* | = | *GEQ* | *GTR* | *NEQ* | ⋕ ⟨ |
⋕ ⟩ | ⟨̲ | ⟩̲ | ≠ | *EQL*;

B.6.2 *Arithmetic Assignment Statements*
⟨ARITHMETIC ASSIGNMENT STATEMENT⟩ ::= *list* [⟨*I⟩
[⋕[⟨SUBSCRIPT LIST⟩ ⋕]]& ←]
⟨ARITHMETIC EXPRESSION⟩;
⟨ARITHMETIC EXPRESSION⟩ ::= ⟨GLOBAL PRIMARY⟩ | ⟨IF CLAUSE⟩
⟨ARITH BOOL EXPRESSION⟩ *ELSE* ⟨ARITH BOOL EXPRESSION⟩ |
⟨ARITH BOOL EXPRESSION⟩;
⟨GLOBAL PRIMARY⟩ ::= [*FOR* (⟨SET VARIABLE LIST⟩) *SIM*
(⟨SET NAME LIST⟩)]& [*SUM* | *PROD* | *GOR* | *GAND* |
MAX | *MIN*] ⟨ARITHMETIC EXPRESSION⟩];

```
⟨ARITH BOOL EXPRESSION⟩ ::= ⟨ARITH BOOL IMPLICATION⟩
                  [EQU ⟨ARITH BOOL IMPLICATION⟩]*;
⟨ARITH BOOL IMPLICATION⟩ ::= ⟨ARITH BOOL FACTOR⟩
                  [[WOR | WEOR | WIMP | WEQV] ⟨ARITH BOOL FACTOR⟩]*;
⟨ARITH BOOL FACTOR⟩ ::= ⟨ARITH BOOL SECONDARY⟩
                  [WAND ⟨ARITH BOOL SECONDARY⟩]*;
⟨ARITH BOOL SECONDARY⟩ ::= ⟨SIMPLE ARITHMETIC EXPRESSION⟩
                  [WNOT ⟨SIMPLE ARITHMETIC EXPRESSION⟩]*;
⟨SIMPLE ARITHMETIC EXPRESSION⟩ ::= [ + | − ]& ⟨TERM⟩
                  [[ + | − ] ⟨TERM⟩ ]*;
⟨TERM⟩ ::= ⟨FACTOR⟩ [[ × | / | DIV] ⟨FACTOR⟩]*;
⟨FACTOR⟩ ::= ⟨PRIMARY⟩ [ ≠* ⟨PRIMARY⟩]*;
⟨PRIMARY⟩ ::= ⟨*I⟩ [≠[⟨SUBSCRIPT LIST⟩ ≠]]& |
                  (⟨ARITHMETIC EXPRESSION⟩) | ⟨MODFUN⟩ |
                  ⟨FUNCTION DESIGNATOR⟩ (⟨ARITHMETIC EXPRESSION⟩);
⟨SUBSCRIPT LIST⟩ ::= list [⟨ARITHMETIC EXPRESSION⟩ |
                  ⟨SET EXPRESSION⟩ | ⟨⟩] separator,;
⟨MODFUN⟩ ::= MOD (⟨ARITHMETIC EXPRESSION⟩,
                  ⟨ARITHMETIC EXPRESSION⟩);
⟨FUNCTION DESIGNATOR⟩ ::= ABS | SIGN | SQRT | TRANS | SIN | COS |
                  ARCTAN | LN | EXP | ENTIER;
```

B.6.3 *Set Assignment Statements*

```
⟨SET ASSIGNMENT STATEMENT⟩ ::= list [⟨*I⟩ ←]
                  ⟨SET EXPRESSION⟩;
⟨SET EXPRESSION⟩ ::= ⟨SIMPLE SET⟩ | ⟨IF CLAUSE⟩ ⟨SIMPLE SET⟩
                  ELSE ⟨SIMPLE SET⟩;
⟨SIMPLE SET⟩ ::= ⟨SET PAIR⟩ [⟨SET PAIR⟩]* |
                  REVERSE ⟨SET PAIR⟩;
⟨SET PAIR⟩ ::= ⟨SET UNION⟩ [PAIR ⟨SET UNION⟩]*;
⟨SET UNION⟩ ::= ⟨SET INTERSECTION⟩ [[UNION | DELETE | SMD |
                  CONCAT] ⟨SET INTERSECTION⟩]*;
⟨SET INTERSECTION⟩ ::= ⟨SET FACTOR⟩ [INTERSECT ⟨SET FACTOR⟩]*;
⟨SET FACTOR⟩ ::= ⟨SET OFFSET⟩ [COMPLEMENT ⟨SET OFFSET⟩]*;
⟨SET OFFSET⟩ ::= ⟨SET PRIMARY⟩ | (⟨SET PRIMARY⟩ [+ | −]
                  ⟨ARITHMETIC EXPRESSION⟩);
⟨SET PRIMARY⟩ ::= ⟨SET IDENTIFIER⟩ | (⟨SET EXPRESSION⟩) |
                  CSHIFT (⟨SET EXPRESSION⟩, ⟨ARITHMETIC EXPRESSION⟩) |
                  ⟨SET DEFINITION TAIL⟩;
⟨SET IDENTIFIER⟩ ::= ⟨*I⟩;
```

APPENDIX C: A METALANGUAGE FOR SPECIFYING SYNTAX

The TRANQUIL syntax in Appendix B is specified in a form of BNF which is extended as follows:[8]

1. Kleene star:
   ⟨A⟩* = ⟨⟩ | ⟨A⟩ | ⟨A⟩⟨A⟩ | . . .
   where ⟨⟩ represents the empty symbol
2. Brooker and Morris' question mark (here &):
   ⟨A⟩ & = ⟨⟩ | ⟨A⟩
3. List Facilities
   list ⟨A⟩ = ⟨A⟩ ⟨A⟩*
   list ⟨A⟩ *separator* ⟨B⟩ = ⟨A⟩ [⟨B⟩ ⟨A⟩]*

4. Brackets
   ⟨T⟩ ::= [⟨A⟩ | ⟨B⟩ | ⟨C⟩] ⟨D⟩
   is equivalent to
   ⟨T⟩ ::= ⟨R⟩ ⟨D⟩
   ⟨R⟩ ::= ⟨A⟩ | ⟨B⟩ | ⟨C⟩
5. Metacharacters:
   A sharp (♯) must precede each of the following characters when they belong to syntactic definitions:
   ♯, [,], *, ;, ⟨, ⟩.

In the syntax ⟨*I⟩ is used to designate an identifier and ⟨*N⟩ is used to designate a number. Further, the special words in the language are *italicized.*

SNAP—An experiment in natural language programming

by MICHAEL P. BARNETT

H. W. Wilson Company
New York, New York

and

WILLIAM M. RUHSAM

Radio Corporation of America
Cherry Hill, New Jersey

INTRODUCTION

Computers are being used to a rapidly increasing extent, to manipulate and to generate materially, mechanically. (1) Many applications simply require items of information to be selected from a file of fixed format, heavily abbreviated records, and expanded into statements that are self-explanatory, and used perhaps in individual communications or incorporated in computer typeset compendia. At the other end of the spectrum are the interrelated challenges of mechanical indexing, abstracting, and translation. The burgeoning applications of computers to publishing, education, library work and information services in most major branches of science and scholarhsip are leading to a host of text processing and generating problems that span these limits of complexity.

Many of the programming problems of mechanical text processing also arise in other kinds of symbol manipulation, such as the compilation of computer programs, and the simplification of mathematical expressions. Several programming languages, such as COMIT and SNOBOL have been developed over the years for mechanical symbol manipulation, and used to process text. FORTRAN, supplemented by some simple assembly language subroutines, has been applied to non-numeric problems quite extensively. ALGOL and COBOL also have been used this way. Several features of PL/1 will facilitate its use in processing text. A number of languages and processors have been developed for special kinds of text processing problems, such as mechanical editing.

SNAP is a procedural language for nonscientists to use. A SNAP procedure consists of a sequence of statements that have the appearance of simple English sentences. The primary rationale of SNAP is that mechanical text processing requires a language that many nonscientists will be willing to learn, and which they will find easy to use. Since such people deal primarily with English sentences in their daily work, a programming language that is a stylized subset of English should seem to them more "natural" than one that is symbolic. This rationale is not negated by the fact that nonscientists are using symbolic languages effectively. Many scientists once learned to write in assembly language, but the number of people programming scientific problems was greatly increased by FORTRAN.

Several SNAP constructions are quite like COBOL. SNAP is designed like BASIC, to enable a beginner to get useful results after learning very little, and to proceed by incremental learning efforts to deal with problems of increasing complexity. In its primary emphasis on string handling, however, SNAP is different; and in the details of the instruction set, and how the basic instructions are expressed, and in the proposed use of statements that invoke subroutines, to allow multitudinous language extensions, SNAP has some novel features.

A prototype processor for the basic SNAP language, that is SNAP without the subroutine capability, was implemented in a compatible subset of FORTRAN IV and now runs on several models of computers.

The language and the prototype processor are used in a graduate course on Computing and Librarianship at Columbia University. An early account of the SNAP project presented the language in a tabular form. (2) SNAP is used as a vehicle to develop basic programming concepts in a recent college text. (3) The implementation of the prototype processor is being reported too. (4) A more advanced processor, that will allow invoking statements and a range of definitional capabilities, is under development.

Programming in MICROSNAP

SNAP statements deal primarily with strings and quantities. A string can be given a name by a SNAP statement such as

```
CALL "YOU CANNOT ERR OR MAKE A
BLOOMER WITH THE WARES OF ANCIENT
SUMER" THE TAG.
```

This statement in a SNAP procedure makes THE TAG connote YOU CANNOT ERR. . until another statement that redefines THE TAG is executed. In general, a CALL statement in SNAP (that is used to give names, *not* to invoke subroutines) consists of (1) the verb CALL, (2) an expression (for example a quotation) that specifies the string which is being given a name, (3) the name, and (4) a period. The forms that are allowed for the expression (2) are described in a later section.

SNAP places almost no restrictions on the choice of words that can be used as names. Extensive restrictions on the choice of names would seriously restrict the ease with which a "natural" programming language could be learned and used, and one objective of the SNAP syntax is to permit meta-words within names. How this may be done is discussed later. For the moment, we just prescribe that a name is a nonblank sequence of letters, digits, spaces, hyphens and apostrophes. Leading and trailing spaces are ignored, so are redundant internal spaces (i.e., any space that follows a space). An article (A, AN, THE) at the beginning of a name is optional (and in fact, ignored, to good advantage—see later).

A string can be printed by a SNAP statement that consists of (1) the verb PRINT, (2) a quotation, or a name that an earlier CALL statement gave to a string, or any of the other forms of string expressions that are to be described in a later section and (3) a period. The one word statement EXECUTE is used to end a procedure, and to make the processor start executing it, so that the SNAP conventions which have been described so far can be demonstrated by typing.

```
PRINT "TESTING, TESTING". EXECUTE.
```

or a little more adventurously, by typing

```
CALL "STILL TESTING" THE MESSAGE.
  PRINT THE MESSAGE. EXECUTE.
```

The computer can be instructed to print output that is longer than the input, ready to be cut into two line display labels for the Mesopotamian Merchants Mart at the Roman Coliseum, by typing as follows:

```
CALL "YOU CANNOT ERR OR MAKE A
BLOOMER WITH THE WARES OF ANCIENT
SUMER" THE TAG. PRINT "CHARIOTS OF
DISTINCTION". PRINT THE TAG. PRINT
"INLAID GAMING BOARDS FOR PORCH
AND PATIO." PRINT THE TAG. PRINT
"DRINKING MUGS FOR THE LONGEST
THIRST" PRINT THE TAG. EXECUTE.
```

The production of display labels that combine a common slogan, or class identification with individual identifications is a very simple and versatile "plot mechanism" for developing examples and exercises that relate to the interests of students in different disciplines. Others include

1. The production of sets of letters that consist of different selections of form paragraphs.
2. The production of programs for several performances of the same play or opera or other artistic work on different occasions, with perhaps some variation of case.
3. The production of a handbill that lists the events for an entire season in which a few different works are repeated many times.
4. The production of messages in large letters made of X's (GO DOG GO, keeps the burden of font design to a minimum), panoramic vistas of seagulls and palm trees on desert islands, processions of stylized animals, etc., printed along the *length* of the output stationery.

More are given in reference 3. The diversity of applications that can be handled with a minimal knowledge of SNAP has just been stressed because is does seem important to enable students of a nontechnical bent to overcome their initial apprehension of the computer by getting results before being overwhelmed by grammatical rules. Once this potential barrier is crossed, most seem able to assimilate programming grammar with ease. The kinds of examples cited here can, of course, be handled with very little grammatical knowledge in many other languages.

Some more verbs that deal with strings

Input statements: The contents of a card, (or its image on magnetic tape, for off-line card input) is immitted by a SNAP statement such as

READ A BIOGRAPHICAL RECORD.

that consists of (i) the verb READ, (ii) the name that the user wishes to give to the string that is read, and (iii) a period. The corresponding instructions that begin with the verbs REQUEST and FETCH immit an input record from the console on which the user signed on (in installations where SNAP is used on line), and from all other input media respectively. In the latter case, the device (and the block format for magnetic tape input) is specified by a statement that begins with the word SELECT, and which remains in force until another SELECT statement that pertains to input is executed. The input commands are being extended to interface with operating system commands, to access files on backing storage conveniently. The REQUEST command alerts the console worker to type a record that is ended by pressing the line feed key; the SNAP processor stores the string under the name that is given in the command, and goes on to the next statement in sequence. There are further SNAP statements to instruct the processor to give certain characters a typographic control interpretation or to use them literally, and to retain trailing spaces in an input record, or to discard them. The latter provision is quite helpful in processes that embed an input item of variable length within a fixed text framework (e.g., a name in a form letter). An input record that consists of several items is deconcatenated by methods that are described later. The name that the input string is given by an input statement may have been used before, but does not need to have been.

Output statements: These consist of a verb, such as PRINT, followed by an expression that specifies the string to be recorded. This expression is, most simply, a quotation, or a name that was defined by an earlier statement in the procedure. Other forms are described later. The verb is PRINT, PUNCH, PERFORATE, TYPE and WRITE for the line printer, card punch, paper tape punch, console and all other media respectively. Instructions that begin with the word SELECT are used to control output on other media in a way that parallels their use for input. The SNAP conventions include provisions for representing case shifts, special characters and font changes in a more limited character set; an appropriate code conversion table can be put into the processor to record output on a device that has an extended typographic capability. A line printer with upper and lower case characters has been driven this way; so has a Flexowriter; and output has been recorded on a magnetic tape that then served as input to the composition programs of the RCA Videocomp electronic typesetting machine.

String synthesis and alteration statements: The CALL statement in general has the form

CALL *b a*.

where *b* denotes a string expression (i.e., an expression that displays or represents a string) and *a* denotes the name that this string is given by the statement. CALL statements are *not* recursive, that is the expression *b* must not make direct or indirect use of the name *a* (the APPEND statement mitigates this—see below). The definition that a CALL statement provides moreover is dynamic, that is, if the interpretation of the expression *b* changes after the CALL statement is executed, then the interpretation of the name *a* automatically changes to correspond. The COPY statement, of the form COPY *b* AND CALL IT *a*. however constructs a copy, in core, of the string that *b* represents, and gives the name *a* to this copy. Subsequent changes in the interpretation of *b* do not affect the interpretation of *a*.

The APPEND statement has the form

APPEND *b* TO *a*.

It gives the name *a* to the result of concatenating the string represented by *b* to the string known previously as *a*.

The OVERWRITE statement has the forms

OVERWRITE *b* ON THE m-th AND (SUBSEQUENT, PRECEDENT) CHARACTERS OF *a*.

Here m-th denotes an ordinal adjective of the kinds such as 3-RD, and UMPTEEN-TH which are discussed later. A single character can be overwritten by a shorter form of the statement.

OVERWRITE *b* ON THE m-th CHARACTER OF *a*.

The DELETE statement, which elides characters, takes the forms

DELETE THE (m-th, m-th THROUGH n-th, m-th AND PRECEDENT, m-th AND SUBSEQUENT CHARACTER(S) OF *a*.

The string name *a* in an APPEND, DELETE or OVERWRITE statement must have been given to a string by a COPY or an input statement previously (and more recently than by any CALL instruction). Expressions that purport to represent strings but which are inconsistent or invalid are considered to represent null strings.

Storage allocation: A SNAP procedure can be written without consideration of the lengths of the strings that are involved, subject to the total core storage capacity of the computer. The prototype processor allows 16K bytes in a 128 K byte RCA Spectra or IBM 360 computer, for strings and internal representations of procedures that are co-resident.

Procedures are condensed appreciably in their internal representation (in "SNAPIC" code), so that for many problems there is ample room for all the strings involved, without recourse to paging tactics; and these can be adopted, using backing storage, when need occurs.

Strings that are defined by CALL statements are represented internally by codes within the actual representations of the statements; but strings which are immitted by input statements, or constructed by COPY, APPEND, DELETE and OVERWRITE statements are stored separately in a "string bank." When a COPY or an input statement is executed, that assigns a string to a particular name for the first time in the current operation of a procedure, this string is stored sequentially in the unused portion of the string bank. When the string that is known by a particular name is changed, the space that it occupies is used for the new string, and chained to a disjoint portion of the string bank if it is inadequate. Since sequentially stored material can be processed more rapidly than disjoint material in many circumstances, SNAP includes an instruction of the form

RESERVE SPACE FOR *n* CHARACTERS IN *a*.

where *n* stands for a positive integer, and *a* for a string name, that may have been used before, but need not have been. This reserves a continuous portion of the string bank, that is *n* characters long, for strings called *a*. It does not preclude longer strings receiving this name—they are simply chained. The statement may be advantageous when the strings that are given the name *a* vary in length during a procedure, and it is possible to anticipate a value which this length is unlikely to exceed, or to impose a limit on this length. Storage allocation thus is permitted to the user, but is not imposed on him.

Instructions that deal with integers

A statement such as

SET I TO 7.

that consists of (1) the verb SET, (2) a mnemonic, word or phrase that the user chooses for a particular quantity, (3) the preposition TO, (4) an integer, and (5) a period, has the dual effect of giving the status of a quantity name to the word(s) or mnemonic that appears between SET and TO, and assigning the value of the integer (item (4)) to this name, until further statements change it. More generally, the item (4) may be either

(i) an integer
(ii) a quantity name that was introduced by an earlier SET statement,
(iii) a length expression such as THE LENGTH OF *b*, where *b* stands for an expression that represents a string (expressions for lengths of lists are discussed later)
(iv) an arithmetic expression of the form

THE f OF e_1 AND e_2

where f denotes one of the words SUM, DIFFERENCE, PRODUCT, QUOTIENT, REMAINDER, CEILING, GREATER, LESSER; e_1 denotes an integer or a name that was introduced by an earlier SET statement, and so does e_2. A name can be used more than once in a SET statement. An article (A, AN, THE) is optional at the beginning of a quantity name and it is ignored when it is included. Invalidity is infectious, that is a quantity defined in terms of an invalid quantity is invalid.
(v) a string expression that represents a string which is a decimal integer, with perhaps redundant leading zeroes, a sign, and leading and trailing spaces.

Defining lists of strings and quantities

A list of strings can be defined by a statement such as CALL "SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY" THE DAY LIST. This permits subscripted names, such as THE 1-ST DAY and THE 5-TH DAY to be used for strings, in any of the ways that unsubscripted string names (e.g., THE TAG used earlier) can be used. A list element thus can be redefined individually, by CALL, COPY, APPEND, DELETE and OVERWRITE statements. It can be recorded by an output statement, and used in expressions that define further strings. A quantity name that has been defined in any of the ways that were described earlier can be used as a symbolic ordinal, by adding -TH, so that if N and UMPTEEN are quantity names, THE N-TH DAY and THE UMPTEEM-TH DAY are acceptable as subscripted string names.

In general, a CALL statement of the form

CALL "s_1, s_2, . . . s_k." THE g LIST.

gives the status of a generic string name to the word or phrase that is denoted by g, and permits expressions of the form THE j-th g to be used as subscripted string names where j-th denotes a numerical ordinal (e.g., 1-ST, 2-ND, 73-RD) or a symbolic ordinal (e.g., N-TH.) A subscripted string name is interpreted as a null string if the ordinal is invalid, or if its value is inappropriate. A comma is forced in a list element by a preceding asterisk, and individual elements are defined to be null by adjacent delimiters (commas and/or quote marks.)

A generic string name also can be introduced by input statements of the form

(READ, REQUEST, FETCH) (A, AN, THE) g LIST.

This immits a record, and treats commas as separators between list elements (except when forced by a preceding asterisk). A generic string name also can be introduced by a statement of the form

RESERVE SPACE FOR k STRINGS IN THE g LIST.

where k denotes a positive integer. The same name can be given to lists of different length, on different occasions in the execution of a procedure, and the processor accommodates these changes automatically by chaining. The RESERVE statement may be used to take advantage of knowledge of a limit that is likely, or which can be imposed.

An entire list can be recorded in the output by a statement of the form

(PRINT, PUNCH, WRITE, PERFORATE, TYPE) THE g LIST.

Successive elements are separated by commas, which are adjacent for elements that are null. Trailing null elements and their commas however are suppressed. Trailing null elements also are ignored in statements of the form

SET k TO THE LENGTH OF THE g LIST.

A further statement, however, of the form

SET k TO THE REGISTERED LENGTH OF THE g LIST.

that includes trailing null elements in the list, as it was defined most recently, will be provided for programmed storage control.

A list of numbers n_1, n_2, . . ., n_k can be defined by a statement of the form

SET THE h LIST TO.n_1, n_2,..., n_k.

This allows expressions of the form THE j-th h (where j-th denotes a numerical ordinal, or a symbolic ordinal derived from an unsubscripted quantity name) to be used as subscripted quantity names, in any of the contexts allowed for unsubscripted quantity names, except symbolic ordinals. RESERVE statements and LENGTH expressions for lists of numbers parallel those for list of strings.

Expressions that represent strings

A string expression, that is an expression which displays or represents a string, may take any of the following forms in a SNAP statement.

1. A quotation, that is bounded by quote marks. Within a quotation, /, \$, 1, >, and < signify forced line break, forced page break, case reversal, upper case and lower case respectively. An asterisk is typed before one of these characters, or a quote mark, to force its literal use within a quotation. Two asterisks are typed to represent a single literal asterisk. An = symbol together with the character that follows represents a special character. When a quotation continues from one input card (line) to the next, one space is included between the last non-blank character on the first card, and the first character on the next, except when the former character is a hyphen, in which case it is elided and the space is not included.
2. An unsubscripted string name, that was introduced by a CALL, COPY, input or RESERVE

statement, at an earlier point in the procedure, both as written and as executed.

3. A subscripted string name that contains (i) a numerical ordinal, or a symbolic ordinal derived from an unsubscripted quantity name that was defined previously; and (ii) a generic string name that was introduced by a CALL, input or RESERVE statement which ends with the word LIST, at an earlier point in the procedure.
4. An integer that is positive, or negative, or zero. Leading zeroes, a plus sign, and spaces before and after the integer in the expression are elided. This is because the integer is first stored as a quantity, and then converted back to a string representation. The statement PRINT 007, thus makes the computer print 7 in the first type position. The statement PRINT "007," however makes the computer print 007 in type positions 1 to 3, since 007 is stored as a string because of the encompassing quote marks.
5. An unsubscripted quantity name that was introduced by an earlier SET statement. This is interpreted, in a context that requires a string expression, as the string of characters that represents the value of the quantity, without any leading spaces, or redundant zeroes, or a sign when it is positive.
6. A subscripted quantity name that contains a generic quantity name which was introduced by an earlier SET . . . LIST . . . or RESERVE statement. This is treated in the same way as (5).
7. An extract expression of the form

   THE k-th CHARACTER OF *a*.

   or

   THE k-th THROUGH j-th CHARACTERS OF *a*.

   where k-th stands for a numerical ordinal, or a symbolic ordinal that contains a previously defined unsubscripted quantity name; and j-th does too, and *a* denotes a string name.
8. A concatenated string expression, that consists of two or more items of the kinds described above, joined by the word THEN, to connote concatenation of the strings that they represent, within the entire string that the expression represents.

Control and conditional statements

A SNAP statement may be preceded by a bracketed label. This label may be cited in control statements of the form

(REPEAT, CONTINUE) (WITH, FROM) g.

where g denotes the label. The verbs REPEAT and CONTINUE are used respectively when g is earlier and later in the written procedure, for external appearances; but they are synonymous as far as the SNAP processor is concerned. The statements

REPEAT FROM THE (BEGINNING, BEGINNING OF THE PROCEDURE).

send control back to the first statement of a procedure, which does not need to be labelled. The one word statement

TERMINATE.

returns control to the operating system.

Two forms of conditional statement are used:

IF u v, OTHERWISE w.
IF u v.

The letters u, v and w here denote the condition, the success action, and the fail action respectively. The success action consists of one or more clauses, that could stand by themselves as *un*conditional SNAP sentences. The clauses are separated by commas, when there are several, and the word AND allowed between a comma and the first word of a clause. The success clause, when there is only one, and the last success clause, when there are several, may be of the (REPEAT, CONTINUE) (FROM, WITH) kind described above. Alternatively, it may be

CONTINUE WITH THE NEXT SENTENCE

This is implied when the success action does not specify a transfer of control.

The fail action may be constructed in just the same ways as the success action, except that

CONTINUE AS FOLLOWS

is used as the final (or only) clause to take the next sentence in sequence. It is implied when the fail action does not specify a transfer of control, and as the entire fail action in the short form IF u v.

SNAP allows the following forms of condition clause at present:

1. THE INPUT IS EXHAUSTED
2. ℓ_1 IS (GREATER THAN, GREATER THAN OR EQUAL TO, EQUAL TO, LESS THAN OR EQUAL TO, LESS THAN, UNEQUAL TO)ℓ_2
3. s_1 (IS, ARE) {THE SAME AS} s_2
4. s_1 IS THE SAME AS THE m-th AND (PRECEDENT, SUBSEQUENT) CHARACTERS OF s_2
5. THE m-th AND (PRECEDENT, SUBSEQUENT) CHARACTERS OF s_1 ARE THE SAME AS s_2.

ℓ_1 and ℓ_2 denote quantity expressions. s_1 and s_2 denote simple string expressions of the kinds (1) to (7) listed previously. m denotes a numerical or a symbolic ordinal that contains a previously defined unsubscripted quantity name.

Some more examples

The account of SNAP in the last few sections covers almost all the features of the basic language. Several of these may be illustrated by the production of a simple calendar of the form

```
WEDNESDAY
   1
  JANUARY
   1969
THURSDAY
   2
  JANUARY
   1969
   :
```

A SNAP procedure to print this is as follows.

```
CALL "JANUARY, FEBRUARY, MARCH,
  APRIL, MAY, JUNE, JULY, AUGUST, SEP-
  TEMBER, OCTOBER, NOVFMBER, DE-
  CEMBER" THE MONTH LIST.

SET THE LIMIT LIST TO 31, 28, 31, 30,
  30, 31, 31, 30, 31, 30, 31.

CALL "SUNDAY, MONDAY, TUESDAY,
  WEDNESDAY, THURSDAY, FRIDAY, SAT-
  URDAY" THE DAY LIST.

SET M TO 1. SET N TO 4. SET K TO 1. SET
  J TO 1.

(PRINT A PAGE ACTION) PRINT THE
  N-TH DAY. PRINT " " THEN J.
PRINT THE M-TH MONTH. PRINT
  "1969/".

IF K IS EQUAL TO 365 TERMINATE,
  OTHERWISE CONTINUE AS FOLLOWS.

INCREASE K BY 1. IF J IS EQUAL TO
  THE M-TH LIMIT INCREASE M BY 1,
  AND SET J TO1, OTHER WISE INCREASE
  J BY 1. IF N IS EQUAL TO 7 SET N TO 1,
  OTHERWISE INCREASE N BY 1. REPEAT
  FROM THE PRINT A PAGE ACTION.
  EXECUTE.
```

This example is quite useful as an illustration of the computers ability to print much more than the user types, by repeating items in different combination. It can be extended and modified in numerous ways, with minimal knowledge of SNAP, which is useful for teaching pruposes.

Another useful example, whose logic is a trifle more elaborate, reads a classification scheme from a deck of cards, such as

```
VERTEBRATA, (MAMMALIA, (PRIMATES,
(ANTHROPOIDEA, (SIMIIDEA, CERCO-
PITHECIDAE, CEBIDAE, H APALIDAE),
LEMUROIDEA, (LEMURIDAE, LORISIDAE,
TARSIIDAE, CHYROMIDAE)), CHIROP-
TERA, (MICROCHIROPTERA, (VESPERTI-
LIONIDAE, RHINOLOPHIDAE, PHYLLOS-
TOMATIDAE), MEGACHIROPTERA, (P
TEROPODIDAE)), INSECTIVORA, ((ERIN-
CEIDAE, TALPIDAE, SORICIDAE, MAC-
:
IIDAE, DIDODONTIDAE, URANOSCOPI-
DAE)))
```

that is strung out to save card space, with brackets indicating subordination, and prints it in a hierarchically indented format, that is

```
VERTEBRATA
 MAMMALIA
  PRIMATES
   ANTHROPOIDEA
    SIMIIDAE
   CERCOPITHECIDAE
  LEMUROIDEA
     :
 CHIROPTERA
     :
```

tor the zoological scheme just cited. The procedure is as follows

```
CALL " " THE BACKGROUND
SET P TO 1. SET Q TO 1.
CALL THE NULL STRING THE CARRY-
  OVER.
(INPUT TEST)
SET J TO O. SET I TO O. SET K TO O.
IF THE INPUT IS EXHAUSTED CONTINUE
  WITH THE LAST ITEM ACTION, OTHER-
  WISE CONTINUE AS FOLLOWS.
READ AN INPUT RECORD,
(CHARACTER TEST)
INCREASE J BY 1. CALL THE J-TH CHAR-
  ACTER OF THE INPUT RECORD THE
  KEY.
IF THE KEY IS "("CONTINUE WITH THE
  DESCENT, OTHERWISE CONTINUE AS
  FOLLOWS.
IF THE KEY IS "," CONTINUE WITH THE
  OUTPUT ACTION, OTHERWISE CONTIN-
  UE AS FOLLOWS.
IF THE KEY IS ")" CONTINUE WITH THE
  ASCENT, OTHERWISE CONTINUE AS
  FOLLOWS.
IF THE KEY IS " " CONTINUE WITH THE
  LAST ITEM ACTION, OTHERWISE CON-
  TINUE AS FOLLOWS. SET K TO J. CON-
  TINUE WITH THE END CARD TEST.
(DESCENT) INCREASE P BY 1. INCREASE
  I BY 1. SET Q TO P.
CONTINUE WITH THE END CARD TEST.
(OUTPUT ACTION) PRINT THE 1-ST
  THROUGH Q-TH CHARACTERS OF THE
  BACKGROUND THEN THE CARRYOVER
  THEN THE I-TH THROUGH K-TH CHAR-
  ACTERS OF THE INPUT RECORD.
SET X TO J. INCREASE X BY 1. SET I TO
  X. SET K TO J. SET Q TO P.
CALL THE NULL STRING THE CARRY-
  OVER. CONTINUE WITH THE END CARD
  TEST.
(ASCENT) DECREASE P BY 1.
(END CARD TEST)
IF J IS LESS THAN 80 REPEAT FROM THE
  CHARACTER TEST, OTHERWISE CON-
  TINUE AS FOLLOWS.
COPY THE I-TH THROUGH K-TH CHAR-
  ACTER OF THE INPUT RECORD AND
  CALL IT THE CARRYOVER. REPEAT
  FROM THE INPUT TEST.
(LAST ITEM ACTION)
PRINT THE 1-ST THROUGH Q-TH CHAR-
  ACTER OF THE BACKGROUND THEN
  THE CARRYOVER THEN THE I-TH
  THROUGH K-TH CHARACTER OF THE
  INPUT RECORD.
TERMINATE. EXECUTE.
```

The procedure can be shortened slightly by omitting OTHERWISE CONTINUE AS FOLLOWS from several IF statements. Hundreds of other examples of SNAP procedures are given in reference 3.

The prototype SNAP processor

The SNAP language was defined almost completely in late 1966. The processor was implemented in stages, in part because there seemed good reasons to demonstrate a working subset, and incremental progress, as quickly as possible, and in part because of uncertainty in the potential of the language, and in some of the details that might be needed. A processor that dealt with CALL and PRINT statements was developed first; input, COPY and unconditional transfer of control statements were added next; then arithmetic operations and conditional statements; and then the further string instructions, and the statements that deal with lists. The prototype processor is written almost entirely in FORTRAN IV. Implementation was started using a time shared PDP 6 computer. After a few months the processor was transferred to an RCA Spectra 70-45, and to the IBM 7094 at the Columbia University Computing Center, where class exercises were run for a semester using the interim version, while implementation was extended on the Spectra. Work on the prototype was ended recently. It is operating at present on several Spectra computers, and on the IBM 360-75/50 system at Columbia University; and a somewhat earlier version compiled and run on a UNIVAC 1108.

The prototype processor consists of (1) a small control section, (2) the translator, and (3) the interpreter. The translator immits a SNAP procedure, and forms a numerical repersentation (in "SNAPIC" code) in the "procedure table". The interpreter then executes the processes that the SNAPIC representation specify. The control section simply calls the translator and the interpreter, and returns control to the operating system when appropriate.

The SNAPIC representation maps a vebral SNAP procedure fairly closely. Integers in different numerical ranges are used for command words, delimiters and precedence codes tor different kinds of expression, and pointers to tables that contain, or point to, objects of interest. An unsubscripted string name in the direct object of a SNAP statement in SNAPIC is represented by a pointer to the "string directory". The corresponding entry in this directory points to a definition of the name in the procedure table, or to the origin of the string in the string bank, depending on whether a CALL or a COPY or input statement that ends with the name was executed more recently. A quantity name is represented by a pointer to the "quantity bank" in which actual numerical values are stored. A subscripted string name is represented by the ordinal (subscript), and a pointer to the string list directory which points in turn to the entry for the first element of the list, in the subscripted string directory. This contains pointers that identify the actual strings which are elements of a list, in the same way that string directory elements identify the strings that are known by unsubscripted names. Pointers to successive elements of a list are stored consecutively in the subscripted string directory whenever possible; chaining is used when necessary. A numerical ordinal is represented by its numerical part, and a symbolic ordinal is represented by the negative of the appropriate pointer to the quantity bank. A subscripted quantity name is represented by the ordinal, and a pointer to the quantity list directory, which points in turn to the origin of the list in the subscripted quantity bank, that contains the actual values of the elements.

The translator was written in an ad hoc fashion. The initial, and incremental capabilities were needed, and obtained, in less time than could be spared to implement a reasonably powerful syntactic analyzer. An analyzer will be used in the translator of the advanced system that is being designed, and additional instructions will be provided to permit users to apply it to data strings at object time.

The control section, translator and processor occupy 56K, 45K and 48K bytes respectively, of which the last two can be overlayed, in the link edited version that runs on the Spectra 70-45. As an indication of the length of SNAPIC representation, the two procedures in the preceding section require just under 600 and 700 bytes respectively.

SNAP as a teaching vehicle

Programming in SNAP can be taught to non-scientists by introducing the basic constructions, and showing some of their uses, in the following sequence.

(1) *MICROSNAP*: This subset of SNAP consists of (i) PRINT and (ii) CALL statements in which strings are displayed as quotations, or referenced by unsubscripted names, and (iii) EXECUTE statements, to start execution. Some exercises for which MICROSNAP suffices were described earlier in this paper.

(2) *MINISNAP*: This subset of SNAP consists of the elements of MICROSNAP and the word THEN. It extends the variety of display labels, form letters, programs for the performing arts and for athletic and sporting events, catalog and greeting cards, and other materials whose mechanized production can be introduced by MICROSNAP, to allow different items to be joined on a line. This has an obvious benefit, for example in the production of form letters.

The addition of the word THEN, moreover, allows the introduction of several relatively general ideas. The production of most hierarchically structured materials (such as a set of catalog cards in which a journal name is repeated throughout while the details of the issue are repeated with infrequent change, and the details of the individual papers are repeated for just a few cards each; or the verbalizations of a long sequence of year numbers) can use hierarchical naming in a variety of ways, particularly when full use is made of the "implied redefinition" characteristic of the CALL statement. Constructing the shortest procedure that is possible for an application, to reduce keyboard work to a minimum, introduces the idea of optimization, in a way that the students can readily appreciate, and which has practical importance when a large volume of material is processed.

The problem of deciding which pieces of the output should be given names during its synthesis, and defining these names most concisely, using just CALL, THEN, quotations and other names, provides a challenge to the ingenuity of the student, after negligible grammatical instruction, that many non-scientists find novel and intriguing. Such examples, moreover, give the student an opportunity to develop an intellectual process, which he can then analyze, and reduce to an algorithm, with the prospect of mechanizing this after learning more programming grammar. An incentive is provided to consider formal descriptions of the structure of strings, which are amenable to simple algebraic manipulations, of potential relevance to the design of large files of data, and to some topics in stylistics, and to learn a little about the elementary uses of graphs.

Some simple combinatorial examples, such as the production of fifteen menus for table d'hote luncheons, that combine one of four appetizers, fish, meat and dessert dishes in all possible ways, can be handled by procedures that are shorter than the output they produce, and which can be generated by procedures that are even shorter still. Such examples help make the student conscious almost from the outset of the course, that procedures can be written to generate procedures, to advantage.

Some mechanized aspects of teaching also can be broached, using MINISNAP. Thus, given an output device with a reasonable typographic capability, the tactics that can be used to print personalized form letters can be applied to the production of a set of texts, that present MINISNAP (or anything else) to reader groups of different professional interests (e.g., middle XVth century armorial bearings, later XVth century armorial bearings, 4.2 mev nuclear physics, 4.25 mev nuclear physics), by substituting examples and exercises of specialized interest in a common explanatory framework. Another educational topic that may merit exploration is the mechanical production of a large number of exercises (from a relatively short prescription) that require the use of different combinations of elementary programming constructions, or mathematical manipulations, or equivalent operational units in other subjects. It is possible that some students might develop their "intuitions" for an activity which involves selecting and manipulating words or symbols, by working large numbers of exercises that could be generated this way, with a saving of human effort, and perhaps analyzed and criticized mechanically. The problem of generating exercises for SNAP starts to approach an interesting level of complexity by the time all the possibilities of MINISNAP are considered. It becomes interesting when the READ instruction is added, which comes next in the progression that is being discussed.

(3) *MIDISNAP:* The addition of the READ statement and REPEAT FROM THE BEGINNING to the elements of MINISNAP permits the separation of procedures and data, and some very elementary data driven procedure generating procedures. Adding extract expressions permits the deconcatenation of input records, and the expansion of fixed field records that omit characters (e.g., decimal points, units of measurement, century digits) which are implicit in the record design, to include these in the output. It is convenient to introduce IF statements that compare strings and statement labels next, then the COPY statement, and then arithmetic. By this stage, additional constructions, and applications are accepted as more or less expected matters of detail. MIDISNAP, that contains the elements which have just been mentioned, is adequate for a considerable range of text generating problems, that use fixed field files on punched cards (or with the addition of SELECT and FETCH, magnetic media); and may well be a convenient subset of SNAP for an elementary course in mechanized text processing and programming concepts, for implementation on relatively small computers.

(4) *Basic SNAP:* The further SNAP constructions that deal with strings and lists open the floodgates of programming rhetoric and applications. Containing these requires a subprocedure capability, such as the one considered next.

Scanning and invocation

SNAP is not the first programming language to be put into use without a subroutine capability, but with the hope that one would be added later. Plans for an extended SNAP processor are well advanced at the time of writing, that will permit subprocedures to be written in SNAP, and invoked by statements that increase the "naturalness" of the language considerably. Quite extensive experimentation probably is needed, with a working system, in working environments, to determine the relative benefits and hazards of the different paths along which the syntax of invoking statements can be developed. The extended processor is being designed to facilitate such experiments, by using a syntax driven translator to produce a canonical representation in extended SNAPIC, for the interpreter to execute.

One line of exploration, that seems to be particularly interesting, would continue to restrict the contexts in which new names for strings, and quantities, and lists, are introduced in procedures and subprocedures. Basic SNAP requires these to be introduced by CALL, COPY, input, RESERVE and SET statements at points which precede their earliest use, in the order in which the procedure is written, to define or alter other strings and quantities. It is possible therefore to scan a CALL statement from left to right alternately for (i) simple string expressions of the forms (1) to (7) given earlier, and (ii) the word THEN, and to treat the balance of the sentence that remains when THEN is not found as the name that the statement confers on a string (or on a list of strings). This name is added to the name table if it has been used before. The tactic can be elaborated, for example to allow names that are substrings of other names. Examples can be constructed to confuse the tactic, probably no matter how much it is elaborated, but the objective of natural language programming is to deal with statements that look "natural" rather than bizarre. The preposition TO can be allowed in the names of quantities (and strings) by using similar tactics in a right to left scan of SET statements.

The tactic can be extended fairly simply to procedures that contain invoking statements of the following kinds:

1. The statement begins with a word that is neither one of the basic verbs of SNAP nor IF; its only arguments are previously defined names, quotations and numbers; the framework in which these are embedded identifies the subprocedure; no

continuous piece of this framework is used as a name in the procedure.

2. The statement begins with a basic verb of SNAP, and contains one or more function expressions, in contexts in which the representation of a string or quantity is appropriate. Each function expression consists of an opening word or phrase that is characteristic of the function, followed by one or more arguments, of the kinds mentioned in the preceding paragraph, in alternation with further words, phrases and/or punctuation that completes the framework which identifies the function. It is convenient to allow either an argument or part of the framework to come last, and to require a comma as the last character in the latter instance. No continuous part of the framework may be a name that is used in the procedure, or a number, and nesting is prohibited.
3. The statement consists of a clause of the form (1) above, a comma, and a further clause such as FORMING X, Y, AND Z CONCURRENTLY, that in general consists of a participle ending in ING, one or more names that need not have been used previously, separated by commas and possibly AND; and a final word (e.g., THEREBY, ACCORDINGLY) to round off the sentence.

Although these conventions are very simple, they cover quite a range of "natural" expressions. The form (1) goes beyond allowing the user a free choice of one word verbs, which often would be insufficient. A verb can be qualified immediately, or at the end of a sentence adverbially or in other ways. Examples of such invoking statements are:

SEND A MEMBERSHIP CARD TO "GRENDEL JONES" AT "THE BEACH HUT".

SEND AN OVERDUE REMINDER TO "DR. FAUST" AT "SANS" SOUCI/BRILLIG DRIVE".

ANNOUNCE "THE CHASED AND THE PURSUER" TO "THE TRASH CAN/75 DREARY LANE" IN SUGGESTIVE TERMS.

ANNOUNCE "THE ILLIBERAL LIBERATION" TO "THE PAPER BACK EGGHEAD/93 FARM YARD" IN INTELLECTUAL UNDERTONES.

which would invoke four separate subprocedures, characterized by the frameworks left by omitting the quotations.

The proposed conventions also allow free use of prepositions (except THEN, which is best left out) in the invoking framework which is a great aid to naturalness. By allowing the generic name of the elements of a list as an argument, the definition of superlative adjectives (as in THE SHORTEST ITEM) is introduced.

The form (1) invoking statement suggested above can be used when a subprocedure does not create any entities for which names do not yet exist in the invoking procedure. Form (2) is useful when one new name must be introduced, and form (3) when several are needed.

A simple convention for heading subprocedures is to begin with a statement of the form PROCEDURE TO followed by the invoking skeleton in which dummy arguments are embedded. These arguments then can be listed in a statement that begins THE ARGUMENTS ARE. .(or THE ARGUMENT IS. .). Function subprocedures can be headed PROCEDURE FOR. .ING. . where . .ING denotes an arbitrary participle (e.g., FORMING, FINDING) and the further dots stand for the function expression with embedded dummy arguments. Some further provisions also will be made for input output, and for user defined conditions.

Introducing the indicative

SNAP, as it has been described so far, consists almost entirely of imperative mood statements. The extended language that is now planned will also include several kinds of indicative mood statements, that will allow statements such as:

(i) THE NAMES OF STRINGS INCLUDE THE SURNAME, THE PRENAME, AND THE ADDRESS.

(ii) THE PRESIDENTIAL RECORD CONTAINS THE SURNAME, THE GIVEN NAME, AND THE DATE OF BIRTH; IN CHARACTER POSITIONS 3 TO 12, 13 TO 25, AND 26 TO 33, RESPECTIVELY

(iii) THE TOWN DATUM LIST CONSISTS OF THE TOWN NAME, THE INCORPORATION DATE, THE POPULATION, THE ALTITUDE, THE MAYOR, THE MAJOR INDUSTRY, THE LARGEST PARK, AND THE MAIN MUSEUM.

(iv) THE OBJECTS INCLUDE THE PRESIDENT, AND THE VICE-PRESIDENT.

(v) THE PRESIDENT IS DESCRIBED BY THE PRESIDENTIAL RECORD.

(vi) THE VICE-PRESIDENT IS DESCRIBED BY THE VICE-PRESIDENTIAL RECORD.

(vii) THE PRESIDENT IS DESCRIBED BY A BIOGRAPHICAL RECORD.
(viii) THE VICE-PRESIDENT IS DESCRIBED BY A BIOGRAPHICAL RECORD.
(ix) A BIOGRAPHICAL RECORD CONTAINS THE SURNAME, THE GIVEN NAME, AND THE DATE OF BIRTH; IN CHARACTER POSITIONS 3 TO 12, 13 TO 25, AND 26 TO 33, RESPECTIVELY.

The sentences (i)-(iii) are, in effect, verbalized forms of type declaration, format statement and equivalence statement. Sentence (iv) is a type declaration that makes THE PRESIDENT and THE VICE-PRESIDENT the names of objects, which sentences (v) and (vi) relate (by a convention that governs the use of IS DESCRIBED BY) to two strings called THE PRESIDENTIAL RECORD and THE VICE-PRESIDENTIAL RECORD. Statements analogous to (ii) could then be used to describe the internal structure of these strings. Statements (vii) and (viii) take a slightly different tack, using a generic string name. By convention these statements would permit the expressions THE BIOGRAPHICAL RECORD OF THE PRESIDENT and THE BIOGRAPHICAL RECORD OF THE VICE-PRESIDENT to be used as string names, for example in input statements; and in conjunction with sentence (ix) would propagate this association, to allow the use of expressions such as THE SURNAME OF THE PRESIDENT to be interpreted correctly. Further simple sentences, that contain the verb HAS, can be used within the framework of some more simple conventions to introduce the names of objects that are attributes of other objects, and described by strings that thereby become indirect attributes of the latter objects.

Syntactic definitions are of considerable importance, and in this regard the kind of meta-syntactic language, and method of representing the result of a syntactic analysis that were developed in the author's laboratory at M.I.T.[5,6] some years ago seem a useful basis for further work.

Many further kinds of definitional device can be postulated that seem potentially useful. For example, class inclusional schemes are given an added dimension by the simple tactic which is illustrated by the following sequence of statements, that relate to a file concerning animals in a zoo.

IN THIS PROCEDURE:
THE KINDS OF CATEGORY INCLUDE SUBKINGDOMS, ORDERS, CLASSES (SINGULAR-CLASS), FAMILIES (SINGULAR-FAMILY), AND SPECIES (SINGULAR-SPECIES).
THE KINDS OF OBJECT INCLUDE ANIMALS.
THE SUB-KINGDOMS OF ANIMALS ARE VERTEBRATES, AND INVERTEBRATES.
THE ORDERS OF VERTEBRATES ARE MAMMALS, BIRDS, REPTILES, AMPHIBIA (SINGULAR-AMPHIBIAN), AND FISH
⋮
THE SPECIES OF APES ARE GORILLAS, CHIMPANZEES, AND ORANG-UTANS.
AN ANIMAL IS DESCRIBED BY A RESIDENT RECORD.
A RESIDENT RECORD CONTAINS THE SPECIES, THE DATE OF ACQUISITION,. . .
⋮
(LOOP START) READ A RESIDENT RECORD. IF THE ANIMAL IS A VERTEBRATE PRINT THE ORDER. . . .

The example has, amongst other things, an element of metonymy. The word SPECIES appears as a kind of category, that includes GORILLAS, CHIMPANZEES etc., and also as the name of a substring of a RESIDENT RECORD. These two uses will be associated, so that when necessary, the contents of the SPECIES field of a RESIDENT RECORD may be compared with the instances of SPECIES in the class inclusional statements. This will make it possible to use the latter words in the data, and to interpret statements such as the IF statement that ends the excerpt.

ACKNOWLEDGMENTS

The work reported in this paper was done in the Graphic Systems Applied Research Laboratory, RCA Laboratories, Princeton, N.J., with which the authors were associated.

REFERENCES

1 *Annual reviews of information science*
John Wiley and Son 1968
2 M P BARNETT W M RUHSAM
A natural language programming system for mechanical text processing
IEEE Transactions on Engineering Writing and Speech
Vol EWS-11 No 2 August 1968 45
3 M P BARNETT
Computer programming in English
Harcourt Brace and World New York Spring 1969

4 W M RUHSAM M P BARNETT
To be published
5 M P BARNETT R P FUTRELLE
Syntactic analysis by digital computer
C A C M Vol 5 1962 515
6 M P BARNETT M J BAILEY
The shadow V system
Unpublished work

The compiled macro assembler

by WARD DOUGLAS MAURER

University of California
Berkeley, California

INTRODUCTION

This paper describes an advance in the art of writing assemblers. It embodies an idea which has been suggested at least twice, but never actually implemented. In a compiled macro assembler, ordinary source language statements are processed in the usual way, but macros are processed in a novel way. The advantage of the compiled macro assembler is the speed with which it processes macros. An actual compiled macro assembler has been written by the author and his students, and the speed with which it processes macros, as distinguished from ordinary statements, has been rigorously tested.

The basic concept of the compiled macro assembler

We review, first of all, the operation of an ordinary assembler, which we will refer to, in what follows, as an *interpreted macro assembler*. (The words "compiled" and "interpreted" are presumed to modify the noun "macro," not the noun "assembler.") Each pseudo-operation code in the assembly language recognized by a given assembler corresponds to a subroutine of that assembler. This subroutine is called whenever the given pseudo-operation is encountered within the source text. The collection of all of these subroutines, for a given assembler, is a fixed collection, and on a large computer this collection of subroutines is normally contained in core at all times. On a small computer, the subroutine which corresponds to a given pseudo-operation may have to be brought in from disk when the pseudo-operation is encountered; however, the total collection of subroutines corresponding to pseudo-operations remains fixed.

A macro is, in one sense, very much like a pseudo-operation. However, in an interpreted macro assembler, the occurrence of a macro does not set aside a special subroutine of the assembler for the use of that macro alone. Instead, all macro definitions are treated in the same way. The text of a macro definition is copied into memory, after various minor transformations such as the removal of blanks. In some assemblers, the information contained in a macro may be further compressed, but in an interpreted macro assembler the compression is done in an essentially recoverable way if it is done at all. When the macro is used, this text is read from memory in what may be called an interpretive fashion—although there is no separate interpreter, the entire assembler itself serving as the macro interpreter.

In a compiled macro assembler, all pseudo-operations—macros as well as others—have their corresponding subroutines of the assembler. At the start of each assembly there exists a fixed collection of such subroutines. However, when a macro is defined, a new subroutine is formed. This subroutine is *compiled* (hence the name, *compiled macro assembler*) from "source text" consisting of the original macro definition. The writing of a compiled macro assembler consists in the mechanization of the process of deducing, from the form of a given macro definition, how a use of this macro would be treated within the assembler if it were a pseudo-operation rather than a macro.

As an illustration of the concept of macro compilation, an actual compiled macro assembler was constructed by the author and his students.* This assembler is written to run on the CDC 6400. The input language is a modified form of IBM 360 assembly language; the output from the assembler is a listing of the IBM 360 code generated, and a deck of binary cards which will execute on the 360 when appropriate control cards are added.

* The students included Donald Alpert, Steven Anderson,[1] Robert Ankerlin, Thomas Baumbach, David Brown, Dennis Griswold,[2] Bing Joe, Richard Kayfes,[3] David Ladd, Kenneth Lew,[4] William Nielsen,[5] Ralph Olstad, Paul Samson, and Edmond Van Doren.[6]

Feasibility of macro compilation

The following paragraphs are devoted to certain feasibility considerations which the author and his students discovered in the course of writing this assembler. These points should be thoroughly understood by anyone intending to write such an assembler in the future.

Substitution of parameters

There are two common methods of handling macro parameters in an assembler. These are known as *string substitution* and *value substitution*. Either may be used in a compiled macro assembler. In addition, if value substitution is used, compilation may be carried out completely; whereas if string substitution is used, it is necessary to include both compiled and interpreted macro facilities, and it may be necessary for a compiled subroutine to call the interpretive facility.

For the sake of completeness, we now describe these two methods in general terms. In value substitution, each actual parameter in a macro usage is evaluated. This value is substituted within the macro text whenever the corresponding formal parameter is encountered. In string substitution, the character string which comprises a given actual parameter in a macro usage is copied into memory when the macro usage is encountered. If the assembler is an interpreted macro assembler, the source of input characters to it is now diverted to the location of the macro text in memory. When a parameter is encountered, the source of input characters is re-diverted to the location of the character string giving the corresponding actual parameter.

String substitution is more general than value substitution because the sequence of input characters passes freely between the characters of the macro and the characters of actual parameters. Thus syntactic units may exist partially within the macro text and partially within the parameter. One important use of this facility is the appending of prefixes or suffixes to an actual parameter to form symbols. If a macro is called with actual parameter DM, for example, the macro may then create symbols DMA, DMB, DM1, TEMPDM, and the like, and use them in an arbitrary fashion. Such symbols, of course, become global, and may be referenced throughout the text. In a value substitution assembler, this facility is not possible; but in many value substitution assemblers a symbol defined in a macro *cannot* be used outside the macro unless it is specially declared to be global. Thus the same symbol may be used over and over again, so long as it is always used inside a macro and only once inside each distinct usage of that macro.

String substitution has been used in most assemblers which have appeared in published work, such as Halpern's XPOP,[7] Strachey's general purpose macro generator,[8] and Mooers' TRAC.[9,10] Value substitution, however, because it is simpler, has been used in many actual, working assemblers. Among these are the FAP assembler for the IBM 7094, the SLEUTH II assembler for the UNIVAC 1107, and an assembler for the UNIVAC III, all of which were written by Ferguson, who, so far as we know, has published only one account of his work.[11]

Let us now consider the substitution of parameters in a compiled macro assembler. If value substitution is used, there is no problem. Suppose that a parameter usage is found within a macro definition. Corresponding to this usage in the compiled subroutine, there is a call to a subroutine which retrieves the value of the corresponding actual parameter. (That is, the compiled subroutine, which is produced by the macro compilation process, calls a fixed, special assembler subroutine, whose function it is to retrieve parameter values.)

If string substitution is used, we make a distinction between a parameter which occurs in an expression in the variable field, and one which occurs by itself in the variable field. (Most actual parameters are of the latter kind, because most people write relatively simple macros.) If a parameter occurs by itself, there is no difference, for this parameter, between string and value substitution, and it may be handled as described above. If a parameter occurs in an expression, however, it is generally impossible to handle it in a compiled manner. The text of the expression must be included with the compiled subroutine, and, at the appropriate point, this subroutine calls a fixed, special assembler subroutine whose function it is to interpretively evaluate such strings. As in the case of an interpreted macro assembler, this "subroutine" consists, from the logical point of view, of the entire assembler itself.

First and second pass compilation

The compilation process, as applied to a macro, must take place twice—once in the first pass and once in the second pass.

There are many reasons for this; the following is perhaps the simplest. Suppose that the definition of a macro involves a symbol which is not defined until after the macro is defined. Then, when the macro is first encountered, complete compilation cannot take place, since the value of the symbol is not known at that time. Therefore the macro must be compiled in the second pass. But it must also be compiled in the first pass, since the length of the generated code is not known, and different uses of the same macro may result in different lengths of generated code.* The main function of the subroutine which is compiled in the first pass, in fact, is to determine this length; at the same time, any global symbols defined within the macro are placed in the symbol table along with their addresses.

It might appear at first sight that this problem could be avoided by defining all symbols used in a macro before the macro is defined. However, this is not feasible in general. A macro may contain a call to an error routine which is at the end of the program, or, in general, which follows another usage of the macro. It is, in general, true that all symbols occurring within a macro definition which affect the length of the generated code must be defined before the macro is defined. By somewhat devious methods this may be improved slightly to read "before the macro is first used."

Saving a compiled subroutine

One of the theoretical advantages in compiling macros is that the resulting compiled code can, in theory, be output to cards, in the same way that output from a FORTRAN compiler can be output to cards. These binary cards may then take the place of the original macro definition.

We have found that compiled subroutines can, in fact, be saved in most cases. There is one case, however, that creates several difficulties. Suppose that a macro definition contains a symbol which is used but not defined. Presumably such a symbol would be defined in the body of the assembly language text. (In our experience, most macros do *not* have this characteristic; but some do, and in any event it would be unwise to exclude it.) The definition of the given symbol in the program in which it is defined is not, however, necessarily the same as its definition in the program in which the binary cards are used. It is this latter definition, in fact, which should apply. Therefore, a distinction must be made when compiling a macro between symbols defined in the macro and symbols defined outside it. There are further difficulties concerned with optimization of the compiled code. If the value of a symbol is known at compilation time, it may be combined with others in an expression, and the value of the result used within the compiled code. If code is being compiled for later use, however, such combination cannot be made. This means that either the resulting compiled code must calculate values of expressions which would not be necessary if the macro were being compiled in that assembly, or the process of loading the binary cards must effectively incorporate some of the compilation process.

Only the second pass compilation need be saved on cards. When this is loaded during the first pass of another assembly, it is loaded in a special way which causes it to act like a first pass compilation.

Compiled macros and conditional and iterative assembly

Conditional statements in assembly language may be compiled; so may iteration statements. In fact, compilation of these statements is the primary justification for compiled macro assembly. A conditional statement in the definition of a macro may be replaced by a conditional transfer in the compiled subroutine; it is no longer necessary to read a number of characters without processing them if the condition is not fulfilled. An iterative (duplication) statement may be replaced by a loop in the compiled code; it is no longer necessary to interpret the iterated statements repeatedly.

A macro which is to be used only once, and which contains no conditional or iterative statements, should not, in fact, be compiled. This is a special case of a general statement which may be made about interpretation/compilation situations: compilation is faster than interpretation only if no recycling takes place. If every statement in a program is to be executed at most once, it is cheaper to interpret each statement once than to compile it (which itself involves interpreting each statement once) and then to execute it. The time saving that results from compiling is due to the fact that if a statement is to be executed several times, it will be interpreted several times if the program containing it is interpreted, but only once if that program is compiled.

A macro without conditional or iterative statements *may* be speeded up on compilation if it is to be used several times, but an intelligent judgment should be made in each such case.

Timing tests of the compiled macro assembler

In order to verify the premise that compiling macros improves the efficiency of macro usage processing, a controlled experiment was performed on the compiled macro assembler written by the author and his students, with the standard IBM 360 F level assembler serving as the control.

* The SLEUTH II assembler embodies an interesting exception to this. If a given macro always generates the same amount of code, this amount may be specified when the macro is defined. Presumably this feature could be implemented in a compiled macro assembler, removing the necessity for compiling such macros on the first pass. However, as we shall see later, such a macro probably should not be compiled anyway.

Timing comparisons of systems designed in different ways to do the same job has proved to be one of the most frustrating tasks in the computing world today. For almost every comparison which has been performed, a perfectly valid argument may be advanced which nullifies its conclusion. Usually this argument takes the form that the observed differences in timing were caused by something other than the differences in the initial conditions. The use of a controlled experiment, a technique borrowed from classical scientific method, is precisely the way in which the effects of such irrelevant factors may be eliminated. In the present situation, the following were the factors which introduced differences in timing comparable to, and sometimes exceeding, the claimed improvements in efficiency:

1. The time taken to process a macro was smaller than the time taken to read a card.
2. The time taken to process a macro was smaller than the time taken to print a line.
3. The total time taken to process a job differed depending on when the job was submitted; in fact, it sometimes happened that when the computer was asked to perform the same job twice in a row (by submitting an input deck consisting of two identical copies of a job deck) the job times differed by a factor exceeding 1.5.
4. The IBM 360 F level assembler as used at the computer center at which the test was made is slower than the Compiled Macro Assembler, by a factor which may exceed 10.
5. The IBM 360 F level assembler is not used at its own greatest efficiency by the computer center at which the test was made.

The controlled experiment was set up in the following way. A macro, RPD3, which generates code to calculate the value of a real polynomial of degree less than or equal to 3, was written for both the Compiled Macro Assembler and the IBM 360 F level assembler. The macro was called, in either assembler, by the line

```
RPD3        X,A,B,C,D
```

where X, A, B, C, and D represent addresses in memory and $A + BX + CX^2 + DX^3$ is the polynomial to be evaluated. The algorithm always uses the fastest computational method; if all of the coefficients are non-zero, then $A + X*(B + X*(C + X*D))$ is calculated, but if any of the coefficients are zero, a smaller amount of calculation is performed. If *all* the coefficients are zero, the result register is loaded with zero. Otherwise, the total number of instructions generated is equal to the total number of non-zero coefficients plus the degree of the largest such coefficient.

A deck was now made up, containing 200 calls to this macro with various parameters. This deck was assembled to obtain a printout of the code it generated. A second deck was now made up which consisted precisely of this generated code. Assembly of these two decks, then, should produce identical results in different ways—with and without macro usage processing. To counteract the effect of factor (1) above, a second macro, called NIL, was written, which does nothing. The text of NIL was added to the first deck, and exactly enough usages of NIL were added to the first deck to equalize the number of cards in the two decks. To be absolutely precise, there were now *four* decks, because all of the above was done twice, once for each assembler.

To counteract the effect of factor (2) above, all assemblies, on both assemblers, were run with a "no list" option during the timing test, after it had been ascertained that they generated correct code. The use of this option insures that no printing will occur during the second pass of assembly. To counteract the effect of factor (3) above, each of these four decks was reproduced several times, and the resulting copies of each deck were run as a connected series of jobs.

The results of the timing test were as follows. For the IBM 360 assembler, the runs without macro calling took 3 min. 21.94 sec., 3 min. 39.92 sec., 4 min. 25.87 sec., and 3 min. 29.00 sec. The runs with macro calling took 9 min. 33.90 sec., 7 min. 52.06 sec., and 7 min. 56.28 sec. Even with the large experimental error, it is clear that this assembler is taking over twice as long to process an assembly with macros as without macros. For the Compiled Macro Assembler, the runs without macro calling took 16.433 seconds and 16.428 seconds; the runs with macro calling took 16.458 seconds and 16.538 seconds. Thus there is no appreciable difference, in the compiled macro assembler, between assembly of macros and assembly of the identical code without macros.

The presentation of the results in this form counteracts factors (4) and (5) above. In particular, any avoidable inefficiencies which affected the timing of one of the IBM 360 runs would also have affected the timing of the other. We also note that factors (1) and (2) do *not*, as has been claimed, remove entirely the timing advantage of compiling macros, since on a time-shared computer the time taken to process a macro will usually *not* be smaller than the time taken to read a card image from a file. It is also true that time-sharing systems increase the viability of assembly language coding as opposed to coding in a higher-level language, since debugging languages (such as DDT and FAPDBG) are much more amenable to machine language than they are to higher level language coding.

ACKNOWLEDGMENTS

The author is grateful for the programming help of the students mentioned in the first footnote to this paper. This research was partially supported by National Science Foundation Grant G-J43 and Joint Services Electronics Program Grant AFOSR-68-1488.

REFERENCES

1 S ANDERSON
Master's report University of California Berkeley January 1968
2 D GRISWOLD
Object deck output from a compiled macro assembler
Master's report Univ of California Berkeley September 1967
3 R KAYFES
Decimal arithmetic in a compiled macro assembler
Master's report Univ of California Berkeley June 1967
4 K M LEW
Non-decimal arithmetic in a compiled macro assembler
Master's report University of Califoria Berkeley June 1967
5 W C NIELSEN
Subsystem implementation of a compiled macro assembler
Master's report University of California Berkeley June 1967
6 E D VAN DOREN
The literal facility and end card implementation of a compiled macro assembler
Master's report Univ of California Berkeley September 1967
7 M HALPERN
XPOP: a meta-language without meta-physics
Proc F J C C 1964
8 C STRACHEY
A general purpose macro generator
Computer Journal October 1965
9 C MOOERS
TRAC, a procedure-describing language for the reactive typewriter
Communications of the Assoc for Computing Machinery March 1966
10 C MOOERS
TRAC, a text-handling language
Proc 20th National ACM Conference 1965
11 D FERGUSON
Evolution of the meta-assembly program
Communications of the Assoc for Computing Machinery March 1966

Some logical and numerical aspects of pattern recognition and artificial intelligence

by W. CLARK NAYLOR

IBM Corporation
Rochester, Minnesota

INTRODUCTION

Artificial Intelligence has received the attentions and contributions of workers in many varied disciplines and of many varied interests. As a result there has arisen a large and diverse body of research literature in the field. The task of sorting out and comparing some threads of continuity through this rich and variegated tapestry presents a tempting prospect.

In this article we define and compare two contrasting pattern recognition approaches. We trace their divergent paths of development from their common origin and emphasize their complementary nature. Finally, we propose their eventual reconciliation and suggest some potentially fruitful lines of development.

Threads of continuity

In 1961 Hawkins[1] examined the state-of-the-art of self-organizing machines and traced some historical developments from early brain models to later computer implementations. This report builds on Hawkins' historical review, emphasizing two separate lines of development and extending them into more recent pattern recognition efforts.

Figure 1 displays two lines of relevant publications by author. At the origin of the lines we indicate the neuron. Section A (below) discusses some of what is known and postulated about the behavior of natural neurons. Section B follows the line of development indicated along the horizontal axis of Figure 1, and emphasizes the logical aspects of pattern recognition. In Section C we follow a line of development displayed along the vertical axis and emphasize the numerical aspects of pattern recognition. In Section D we discuss some of the problems of dealing with both aspects of the pattern classification problem at once.

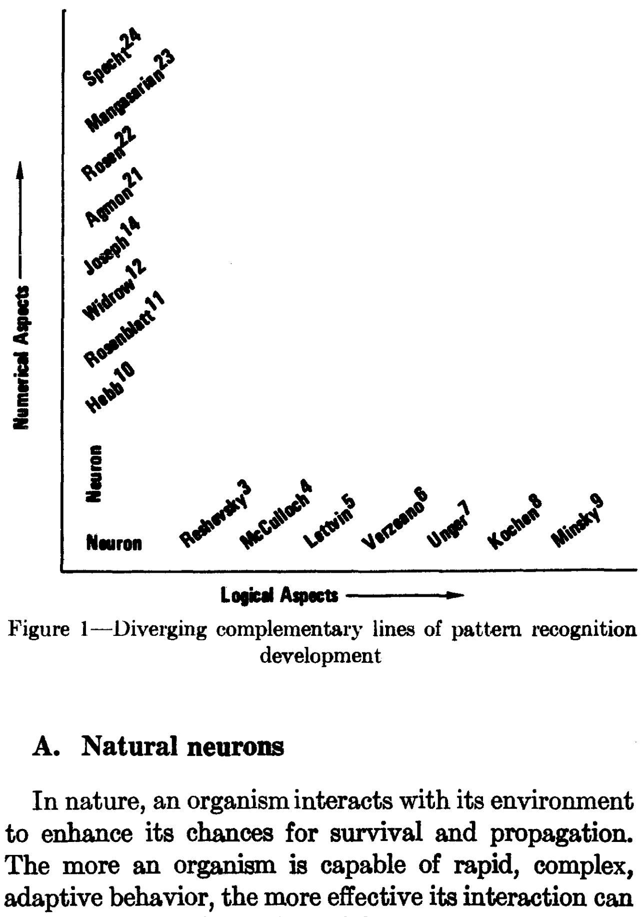


Figure 1—Diverging complementary lines of pattern recognition development

A. Natural neurons

In nature, an organism interacts with its environment to enhance its chances for survival and propagation. The more an organism is capable of rapid, complex, adaptive behavior, the more effective its interaction can be. In the animal world special sensors, effectors, and associated nervous systems have developed to achieve this rapid, complex behavior. And although the complexity of the nervous systems varies greatly from the lowest to the highest animals, the properties and behavior of the basic nerve cell, the neuron, remain amazingly constant.

The neuron is a cell specialized for conducting electrical signals. It is the cell of which all nervous systems are constructed. In vertebrates, bushy dendrites extend from the cell body to receive afferent excitation and conduct it to the axon. The axon, on receiving sufficient excitation, "fires" and conducts a spike pulse along its length to the axonal branches. There the excitation is communicated across various synaptic junctions to succeeding dendrites, and so on. After firing, the cell enters a refractory state during which it rests and recharges its membranes in preparation for the next firing. Some neurons can repeat this cycle hundreds of times a second.

Many neuron configurations exist. Neurons may have long axons, very short axons, several axons, or no apparent axons at all. They may have many dendrites or no descernible dendrites. Dendrites and axons may be virtually indistinguishable. Likewise, many varieties of synapses exist. Some transmit electrically, some transmit chemically. Some transmit axon-to-axon and some transmit dendrite-to-dendrite. Probably the most interesting are the synapses between axonal branches and soma or dendrites, typical in vertebrate brain cells.

B. Logical development

A network of axons, each capable of a binary all-or-none response, is strongly suggestive of switching theory and logic, and much of the work in pattern recognition and artificial intelligence is based on this observation.

Rashevsky[3] in 1938 was perhaps the first to postulate that nets of such binary axons could perform certain decision and memory functions. McCulloch and Pitts[4] in 1943 formalized these concepts and showed that the behavior of such nets could be described in terms of Boolean algebra. Later, Lettvin, Maturana, McCulloch, and Pitts[5] in 1959, and Verzeano and Negishi[6] in 1960, were able to experimentally substantiate some of these ideas.

In 1959, Unger[7] described a pattern recognition program in which he used the logical structure of a binary tree to separate an alphabet of 36 alphanumeric characters. In 1961, Kochen[8] described a concept formation program which could adaptively derive its own Boolean sum of products from its experience with the data. And, in 1967, Minsky[9] considered general machines composed of McCulloch-Pitts neurons. He established several universal bases for building finite automata, and showed that a very simple "refractory cell" formed such a base.

C. Numerical development

While the logical development exploits the logical ability of the axon behavior, it greatly oversimplifies or largely ignores the role of the synapse.

In 1949, Hebb[10] suggested that perhaps the synapse provided the site for permanent memory. He postulated that the ability of the axonal branches and the dendrites to form graded potentials, and the ability of the synapse to differentially attenuate and integrate the influence of many impinging signals, might somehow change as a function of learning. In 1958, Rosenblatt[11] incorporated these and other ideas into a model he called the Perceptron. At about the same time, Widrow[12] began experiments with similar analog models he called Adalines. Many workers[13–21] showed the ability of these models to implement linear decision surfaces, and the ability of certain training procedures to converge to *feasible* surfaces. In 1963, Rosen[22] employed quadratic programming to obtain *optimal* decision surfaces for both linear and quadratic models. In 1964, Mangasarian[23] obtained optimal decision surfaces using linear programming. Based on a Bayesian statistical approach, Specht,[24] in 1967, derived optimal decision surfaces for general n^{th} order polynomials.

D. Combined logical and numerical aspects

In his critical review of Artificial Intelligence work, Dreyfus[29] addressed himself primarily to workers specializing in logical methods. In criticizing the assumptions of Newell, Shaw, and Simons[30] he said "they do not even consider the possibility that the brain might process information in an entirely different way than a computer—that information might, for example, be processed globally the way a resistor analogue solves the problem of the minimal path through a network." In the present context, this and other similar comments in his paper seem to be suggestions for more careful consideration of numerical, as well as logical, methods.

While it appears that some workers have been applying logical tools to geometrical tasks, it also appears that other workers have been applying geometrical tools to logical tasks. For example, in the layered machines mentioned in Nilsson,[25] it is necessary for the first layer of linear decision surfaces to completely partition the input space. Succeeding layers of linear decision surfaces then operate on the output of previous layers and so on. However, when the input space has been completely partitioned, it has been mapped without conflict onto the vertices of a hypercube. When this has been accomplished, only a problem in Boolean logic remains, and it seems a little wasteful to use additional layers of linear decision surfaces for this task. Moreover, no general training procedures for such machines have yet been found.

While those workers mentioned in Section B have had success in dealing with the logical aspects of the pattern recognition problem, and those workers in Section C have had success in dealing with the numerical aspects, few workers have been successful in dealing with both aspects at once. However, some recent approaches appear very promising in this direction. In 1965, Casey[26] described a program for reducing the dimensionality of sets of patterns. This would appear to be a good first step toward discovering the structure of a problem. Ball and Hall's program[27] to automatically cluster sets of patterns can be viewed as a process for finding logical structures of imbedded numerical decision surfaces. The most clear-cut example in this direction is the 1968 program of Mangasarian.[28] This program iteratively builds a ternary tree of linear decision surfaces. Each surface is designed to be the optimum for its level on the tree, and the tree is indefinitely expandable.

The complementary nature of logic and geometry in pattern recognition

We would like to argue in the following sections that the two divergent lines of development pursued in the previous sections are not alternate approaches to the same problem but rather complementary approaches to that problem. That is, that a general approach must involve both aspects and that an approach emphasizing only one aspect must be somehow incomplete. This argument must be based on efficiency rather than ultimate effectiveness since either approach may be employed to eventually obtain a very good approximation to the desired result.

A. Set theory and pattern recognition

If we view pattern recognition in a set theoretic framework, the roles played by the two ordering relations, set membership, $\in$, and set inclusion, $\subset$, are very significant. If we are dealing with sets (or patterns) of real vectors, we see that we have two distinct algebras involved.

Among the *sets* themselves, we have the algebra of set combination or logic. Among the *members* of the sets, we have the algebra of real numbers, arithmetic or geometry.* The difference in emphasis evident in the logical and numerical developments amounts to a difference in emphasis on the roles of the two algebras involved. Thus, in the logical development, the ultimate classes are composed of complex combinations of sets with very simple membership criteria. The algebra of set combination (viz logic) is strongly emphasized, while that holding among set members (viz geometry or arithmetic) is largely ignored. On the other hand, in the numerical development the ultimate classes have no apparent constituent sets and the criteria of set membership must bear the whole burden of the classification task. Thus the algebra holding among the set members (viz arithmetic or geometry) is strongly emphasized while the algebra among the sets is largely ignored.

* In some examples of pattern recognition we may have other relations holding among set members (for example, grammatical relations in language translation). If the algebra among set members happens to be Boolean logic, then this whole distinction may disappear.

B. Example

The complementary nature of the logical and numerical approaches may be likened to the complementary nature of Fourier and polynomial series. The Fourier series may be used to approximate a straight line, and the polynomial may be used to approximate a sine curve, but it is an unnatural and wasteful way to use the series. Similarly, logic may do the job of geometry or geometry may do the job of logic, but it is wasteful not to put each technique to its natural use. A simple example will illustrate that sets which are simply and naturally described in terms of logical combinations of numerically defined constituent sets may be very difficult to describe by logic or geometry alone. Consider the sets of rectangles defined as follows:

A: Circumference less than 20 units
B: Area less than 9 units
C: Area more than 4 units
D: Vertical side no shorter than 1/2 the horizontal side
E: Horizontal side no shorter than 1/2 the vertical side
F: $((B \cap C) \cup (D \cap E)) \cup A$

The set F is simple enough. Try to describe it by geometry or logic alone! Figure 2 is a sketch of set F.

C. A proposed response surface

The idea of the complementary nature of logic and and geometry is simple enough. Is it possible to quantify it and illustrate it graphically? Consider the following proposed axes:

X. Average number of members per component set

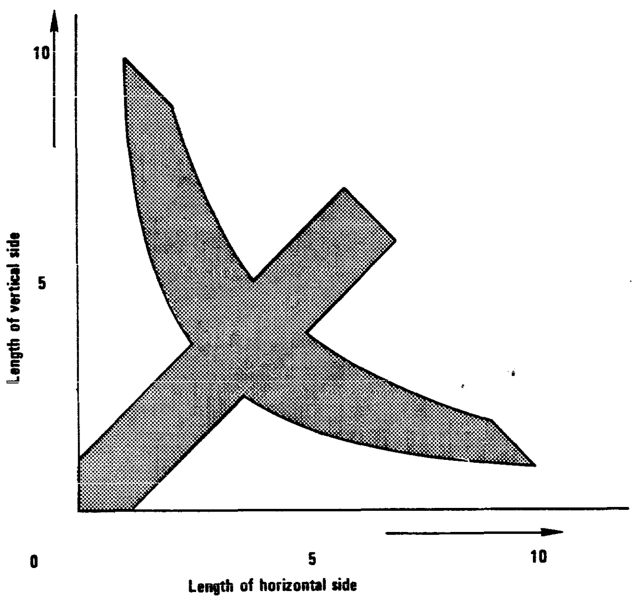


Figure 2—The set F

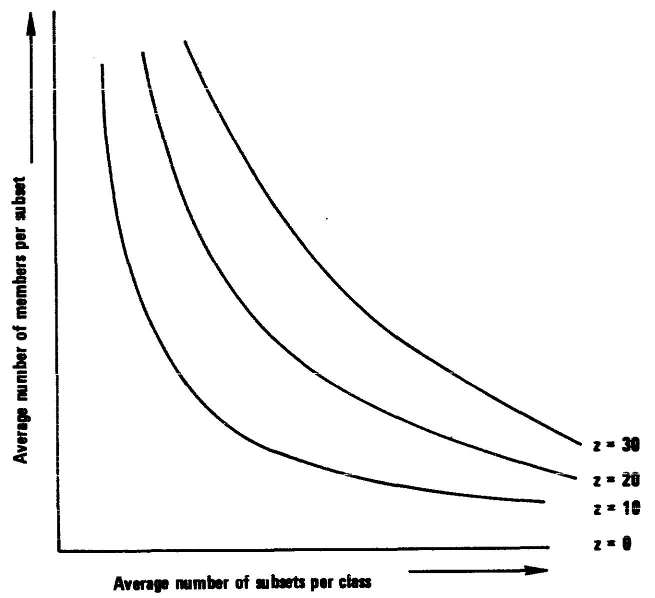


Figure 3—Suggestive sketch of proposed response surface

Y. Average number of component sets per pattern class

Z. Percentage of correct recognitions achieved

If we classify various pattern recognition programs as (X, Y) points and plot Z(X, Y) for each program, what sort of graph would result? Obviously one contour must be Z(0, Y) = Z(X, 0) = 0. If we further assume Z to be continuous and monotonic then contours such as those of Figure 3 will result. Figure 4, showing the relative paths of logical and numerical developments on such a surface, illustrates graphically the relative performances of logical and numerical methods.

Some optimality criteria

Having divided the pattern recognition methods into logical and numerical classes, we will find it useful and interesting to further subdivide the numerical class according to the optimality criteria used.

In numerical analysis, if we are trying to obtain the best fit of a line to a set of points, we generate an error vector and attempt to minimize some norm of the vector. The P-norm of a vector y = $(y_1, y_2 \ldots, y_n)$ given by:

$$L_P = \left[\sum_{i=1}^{n} |y_i| P\right]^{1/P}$$

is the norm most commonly used for this purpose.

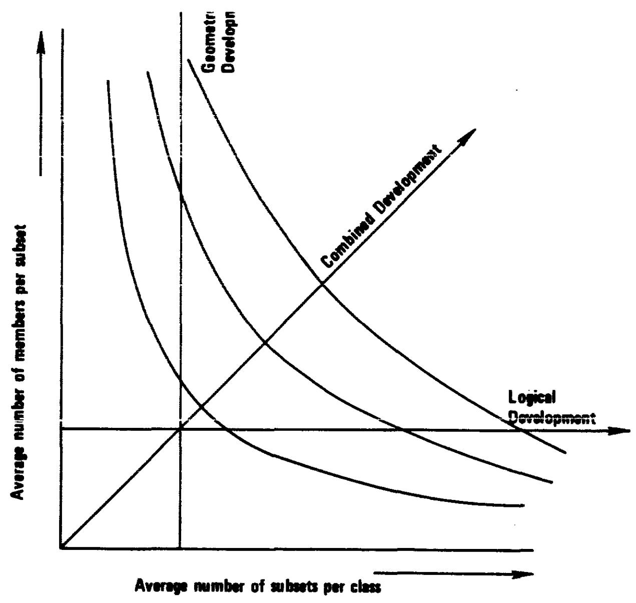


Figure 4—Suggestive sketch of various development paths

The values of P commonly used are P = 1, P = 2 and P = ∞. For P = 1, we minimize the average error. For P = 2, we minimize the sum of squares error. For P = ∞- we minimize the maximum error (Chebyschev criterion).

In pattern recognition we have a very similar situation. For any separating surface we generate a vector of separation distances and attempt to maximize the overall separation. In analogy with the one norm, we may attempt to maximize the average separation; in analogy with the Chebyschev ∞ norm, we may attempt to maximize the minimum separation; or in analogy with intermediate norms we may similarly choose a whole spectrum of optimality criteria.

It is instructive to consider the sensitivity and stability of methods employing the two criteria on the extremes of this spectrum. On the one hand, a method to maximize the minmum separation will seek out the few "worst-case" points and work on them first. Such a worst-case method will be a local, differentiative method; it will be very sensitive to local details, but very prone to over-react to noise. On the other hand, an average-case method will be a global, integrative method. It will tend to be relatively insensitive to noise, but also insensitive to local detail.

An example will illustrate this noise and detail sensitivity. Consider the sets and separating plane of Figure 5. The plane satisfies worst-case, average-case and intuitive criteria for a good separating plane.

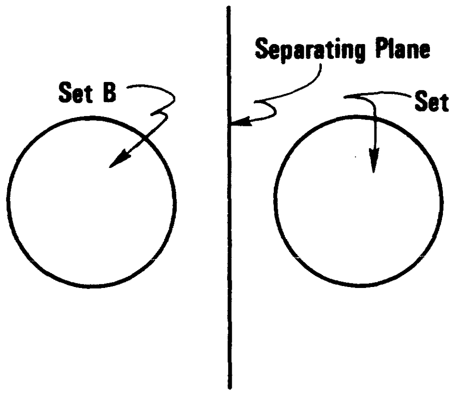


Figure 5—Sets and separating plane

Consider the sets and planes of Figure 6. Here set A has been augmented by A_2. As long as the minimum difference between points in B and A_2 is larger than the minimum difference between B and A_1, A_2 will have no effect on the placement of plane 1, the worst-case plane. The A_2 information is essentially redundant. However, plane 2, the average-case plane, will move around and and react to set A_2 as set A_2 moves. It may even violate one of the sets. Intuitively we would probably choose plane 1.

Consider the sets of planes of Figure 7. Here Sets A and B have been augmented by noise patterns. Since these noise patterns affect the minimum distance between the sets, the worst-case plane, plane 1, will react, while the average-case plane, plane 2, does not. Here we would probably intuitively choose plane 2.

Again we can tie these items to physiological considerations. Several averaging and differentiating neuronnets have been observed in nature[6,7]—particularly in optic nerves. Apparently their actions are carefully balanced to insure sharp resolution together with noise

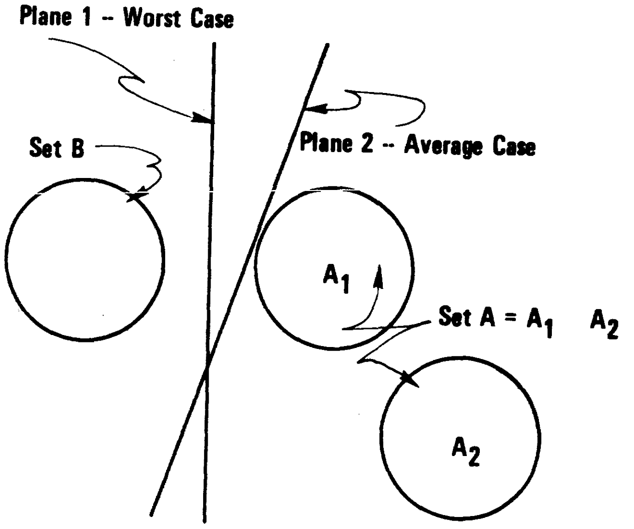


Figure 6—Multimodal sets and separating planes

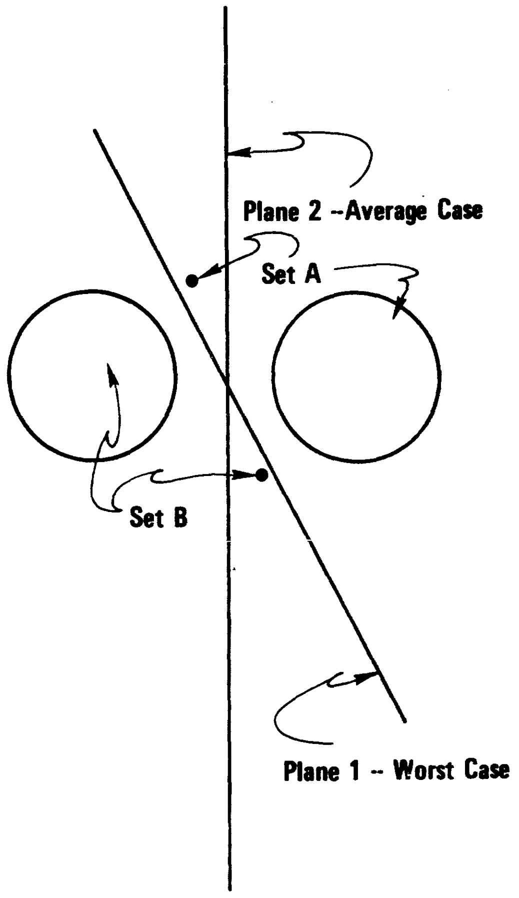


Figure 7—Sets and separating planes with noise

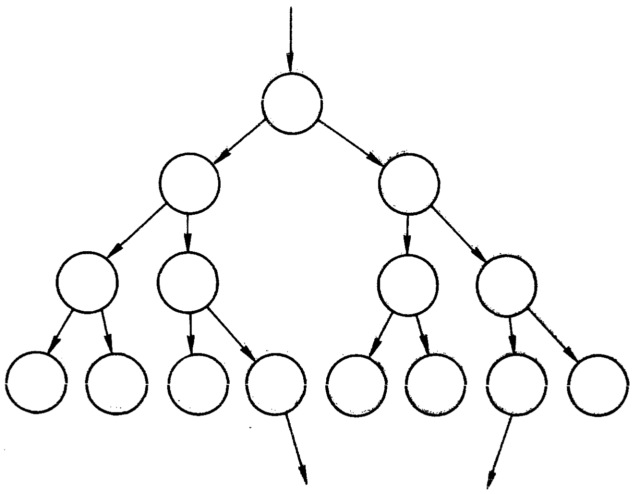


Figure 8—Proposed pattern recognition structure (shown for binary tree as an example)

insensitivity. In pattern recognition we will similarly be forced to achieve a balance in the use of average and worst-case methods.

Proposed model for future development

From the considerations of this report, it seems clear that a general pattern recognition device will have to perform numeric calculations carefully balanced between global integrative techniques and local differencing techniques. The results of these calculations will then be combined logically to determine the result of the entire device. It also seems clear that natural nervous systems will provide an existence proof and guide in the construction of feasible pattern recognition models.

Figure 8 represents a possible logical-numerical net. At each node a numeric calculation is performed and an exiting branch is chosen. The overall branching network of nodes provides the logical structure (although a logical tree is shown for simplicity, any complex logical structure is intended).

SUMMARY

Pascal once said that there are two kinds of mathematical minds—logicians and mathematicians. We have indicated that there are two kinds of pattern recognition programs, logical ones and geometrical ones. In this report we have traced the historical development of these two distinct approaches. We have related them to two functions of natural nerve nets and to the two algebras of set theory. By these associations we have argued for the complementary nature of the roles played by these two aspects. In addition we have distinguished and compared the roles of average-case and worst-case geometrical methods. Finally, for future development, we have suggested a pattern recognition model encompassing the capabilities of all these methods.

ACKNOWLEDGMENTS

Work leading to this report was carried out in the Computer Science Department of the University of Wisconsin and the Recognition Systems Development Department of IBM Rochester, Minnesota. The author is indebted to many people for their help and encouragement during this work: Drs. L. Uhr, J. B. Rosen, and O. L. Mangasarian at the University of Wisconsin for many stimulating and enlightening discussions; A. Hamburgen, D. R. Andrews, P. H. Howard, and M. J. Kimmel of the Recognition Systems Development Department at IBM for their continued encouragement and support; and C. D. Cullum of IBM Yorktown for his helpful ideas.

REFERENCES

1 J K HAWKINS
Self-organizing systems—A review and commentary
Proc of Institute of Radio Engineers January 1961

2 T H BULLOCK A G HORRIDGE
Structure and function in the nervous systems of invertebrates
W H Freeman and Company New York N Y 1965

3 N RASHEVSKY
Mathematical biophysics
University of Chicago Press Chicago Ill 1938

4 W S McCULLOCH W H PITTS
A logical calculus of the ideas immanent in nervous activity
Bulletin of Mathematics Biophysics Vol 115 1953

5 J Y LETTVIN H R MATURANA
W S McCULLOCH W H PITTS
What the frog's eye tells the frog's brain
Proc of Institute of Radio Engineers Vol 47
November 1959 1940–1951

6 M VERZEANO K NEGISHI
Neuronal activity in cortical and thalamic networks
J Gen Physiol Vol 43 supply July 1960 177–195

7 S H UNGER
Pattern detection and recognition
Proc of Institute of Radio Engineers October 1959

8 M KOCHEN
An experimental program for the selection of disjunctive hypothesis
Proc Western J C C Vol 19 571–578 May 1961

9 M MINSKY
Computation: Finite and infinite machines
Prentice Hall Englewood Cliffs New Jersey 1967

10 D O HEBB
Organization of behavior
John Wiley and Sons Inc New York N Y 1949

11 F ROSENBLATT
The perceptron—A theory of statistical separability in cognitive systems
Cornell Aeronatuical Lab Buffalo New York Report No VG–1196–G1 January 1958

12 B WIDROW M E HOFF
Adaptive switching circuits
Stanford Electronics Lab Stanford California Technical Report No 1553–1 June 1960

13 F ROSENBLATT
On the convergence of reinforcement procedures in simple perceptrons
Cornell Aeronautical Lab Report No VG–1196–G4 February 1960

14 R D JOSEPH
Contributions to perceptron theory
Cornell Aeronautical Lab Report No VG–1196–G7 Buffalo N Y June 1960

15 H D BLOCK
The perceptron: A model for brain functioning–I
Reviews of Modern Physics Vol 34 123–135 January 1962

16 A CHARNES
On some fundamental theorems of perceptron theory and their geometry
Computer and Information Sciences Spartan Books Washington D C 1964

17 A B J NOVIKOFF
On convergence proofs for perceptrons
Stanford Research Institute Report Nonr 3438(00) January 1963

18 R C SINGLETON
A test for linear separability as applied to self-organizing systems—1962
Spartan Books 503–524 Washington D C 1962

19 W C RIDGEWAY
An adaptive logic system with generalizing properties
Stanford Electronics Lab Technical Report No 1556–1 Stanford University Stanford California April 1962

20 T S MOTZKIN I J SCHOENBERG
The relaxation method for linear inequalities
Canadian Journal of Mathematics Vol 6 No 3 393–404 1954

21 S AGMON
The relaxation method for linear inequalities
Canadian Journal of Mathematics Vol 6 No 3 383–39 21954

22 J B ROSEN
Pattern separation by convex programming
Journal of Mathematical Analysis and Applications Vol 10 No 1 February 1965

23 O L MANGASARIAN
Linear and nonlinear separation of patterns by linear programming
Operations Research Vol 13 No 3 May 1965

24 D F SPECHT
Generation of polynomial discriminant functions for pattern recognition
Stanford Electronics Lab Technical Report No 6764–5 Stanford University Stanford California May 1966

25 N NILSSON
Learning machines
McGraw-Hill Inc New York N Y 1965

26 R G CASEY
Linear reduction of dimensionality in pattern recognition
IBM Research Report R C–1431 Yorktown Heights N Y March 19 1965

27 G H BALL D J HALL
ISODATA, an iterative method of multivariate analysis and pattern classification
Proc of International Communications Conference Philadelphia June 1966

28 O L MANGASARIAN
Multi-surface method of pattern separation
(to be published)

29 H L DREYFUS
Alchemy and artificial intelligence
The RAND Corporation Santa Monica California December 1965

30 A NEWELL H H SIMON
Computer simulation of human thinking
Science Vol 134 2011–2017 December 22 1961

31 I P PAVLOV
Conditioned reflexes
Oxford University Press New York N Y 1927

32 D A SCHOLL A M UTTLEY
Pattern discrimination and the visual cortex
Nature 387–388 February 28 1953

33 W A CLARK B G FARLEY
Generalizations of pattern recognition in a self-organizing system
Proc W J C C 86–91 1955

34 S B AKERS
Techniques of adaptive decision making
General Electric Company Electronics Laboratory Technical Information Series R65ELS–12 Syracuse New York October 1965

35 G L NAGY
Prospects in hyperspace: State of the art in pattern recognition
IBM Research Paper RC–1869 IBM Watson Research Center Yorktown Heights New York June 1967

36 M D CANON C D CULLUM
The determination of optimum separating hyperplanes I. A finite step procedure
IBM Research Report RC–2023 IBM Watson Research Center Yorktown Heights New York June 1967

37 L UHR
Pattern recognition
John Wiley and Sons Inc New York N Y 1966

38 G S SEBESTYEN
Decision-making processes in pattern recognition
Macmillan Company New York N Y 1962

A model of visual organization for the game of GO

by ALBERT L. ZOBRIST

University of Wisconsin
Madison, Wisconsin

INTRODUCTION

No successful GO-playing program has appeared in the literature, although Remus[1] used GO as the subject of a machine learning study, and Thorp and Walden[2] have considered some of its mathematical aspects. Another author[3] considered GO to be somewhat mysterious, making it a challenge to those interested in automating it. Apparently the game was described as being mysterious to indicate that people were able to play it without knowing how they were able to play so well. More study of this complex game may reward us with new insight into human perceptual and problem solving abilities as well as foster the development of new techniques for artificial intelligence. This report describes a program which plays GO. The program uses an information processing model to produce perceptual features which are seen by human GO players, and is capable of several responses to the recognition of significant configurations of these perceptual features.

A brief description of GO

The rules of GO are deceptively simple. The two players alternate in placing black and white stones on the intersections of a 19 × 19 grid. Stones of the same color which are connected by row or column adjacency form a *chain*. Diagonal adjacency is not sufficient to connect a chain. The empty intersections which are adjacent to a chain are its *breathing spaces*. When a chain has no breathing spaces, it is captured by the opponent, and the captured men are removed from the board. A player may place his stones anywhere on the board with two exceptions: (1) he may not form a chain with no breathing spaces unless he is capturing, and (2) he may not capture one stone which has just captured one of his stones on the previous turn. A player may choose to pass at any turn. The game is over when both of the players pass in sequence. A player's score is the sum of territories surrounded by his color plus the number of opponent's stones captured.

Some of the basic consequences of these rules are illustrated by the right side of Figure 1. White can always capture the top black chain, but cannot capture the bottom black chain, if black moves Figure 1 properly. If black moves at either of T2 or T3 then white cannot occupy all of the breathing spaces of the black army without committing self-capture. This is because the black army would have two separate *eyes*. The ability to form two eyes is what determines whether an army is safe or not. White will score 16 points in the upper right, and black will score four points in the lower right corner. If white moves R10, then black may not respond R11, but must move elsewhere on the next turn. This prevents cyclic capture.

The rules scarcely describe how GO is actually played. Interested readers are advised to seek a demonstration from someone who plays GO, or to read one of the beginner's books.[4,5] The situations in the left hand corners of Figure 1 are representative of real play. Although the stones are not connected into long chains, they threaten to form chains which will surround territory along the corners and edges of the board. Efficient play requires that as much territory be sketched out with as few stones as possible. Throughout the rest of this paper such aggregates of stones which threaten to invade or surround territory will be called *armies*.

The problem of complexity

GO is considered more difficult than chess by many people who know both games.[5] Numerical measures of the complexity of checkers, chess, and GO tend to support this belief. The number of paths down the move

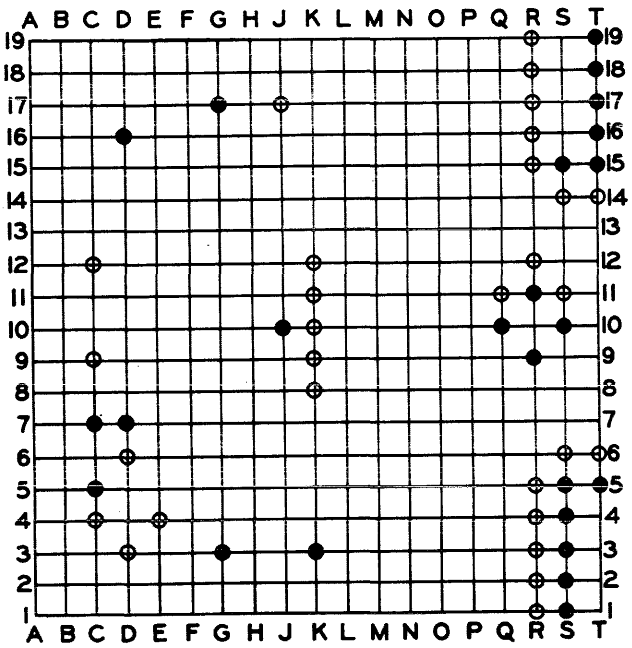


Figure 1—An illustration of GO

tree has been estimated at 10^{40} for checkers[6] and 10^{120} for chess.[7] A rough estimate for the number of paths down the move tree for GO is 361! or 10^{761}. By this reasoning, GO played on a 6 × 6 board would be comparable to checkers in complexity, and GO on a 9 × 9 board would be comparable to chess.

A slightly better extimate of the true complexity of these games may be obtained. For checkers, suppose that a choice of three reasonable moves occurs approximately 20 times per game. Then 3^{20} is a crude estimate of the number of checker games which might occur in ordinary play. Good chess players usually consider less than five move choices, hence 5^{50} estimates the number of reasonables chess games. A typical GO game lasts about 300 moves and a choice of 10 reasonable moves occurs at least 100 times, thus there are at least 10^{100} GO games which could occur in humn play.

Such calculations, however crude they may be, are important to anyone interested in the automation of these games. The complexity of GO may hinder attempts to program it with the methods developed for chess and checkers.[6,7,8,9] The move tree for GO is exceedingly deep and bushy, hence any form of heuristic search can explore only a relatively small portion of the complete tree. An alternative approach might be to concentrate upon extremely powerful methods of evaluation of the board situation, thus enabling better play with a more restricted search. Another possibility might be to have the lookahead be directed by pruning methods which correspond to the development of strategies. Time will tell whether a successful GO playing program can be written using such methods.

The visual nature of GO

The recognition and discrimination of meaningful perceptual stimuli presupposes the active formation of stable perceptual elements to be recognized and discriminated. A person lacking this process would combine all sorts of stimuli into meaningless groups.[10] The choice of a move in GO usually involves the recognition of configurations which are meaningful to the player. This raises the question as to whether certain basic perceptual processes are necessary for the comprehension of a GO board. The following examples might suggest that the answer is yes.

First, consider the spontaneous grouping of stones of the same color which occurs during visualization of a GO board. The stones are organized into distinct groups, clusters, or armies even though they may be sparsely scattered about or somewhat intermingled. Grouping is usually the result of proximity of stones of the same color or the predominance of stones of one color in an area, but can be affected by other characteristics of the total board situation. For example, stones which fall into a line are likely to be grouped. Kohler[11] and others have found grouping to be a basic perceptual phenomenon. Yet the recognition and discrimination of groups or armies is necessary for competent GO play.

Closely related in grouping is segmentation, which is also discussed in Kohler. The area subtended by the board is divided into black and white territories, each of which maintains its own integrity in the visual field. These segments are a measure of the territory which is controlled by either side, hence are an important factor in the assessment of a GO board.

Another example is the formation of "spheres of influence" about a stone or group of stones. Influence is not an inherent property of stones, but appears to be induced in them by our processess of perception. Yet they are a crude measure of the potential of a stone or army of stones for controlling territory on the board.

The spontaneous image formed by the visualization of a GO board appears to be a complicated assemblage of perceptual units and subunits. For example, the stones themselves have their own perceptual identity while at the same time they are parts of chains or groups of stones. The phenomena discussed above show that some of these perceptual processes may be very important to the ability of GO players to comprehend this complex game.

It is not within the scope of this report to discuss further the psychological nature of these perceptual mechanisms, or to speculate upon the physiological basis for them. Let us adopt the term *visual organization* to mean the formation of such stable perceptual elements as have just been discussed, and let the resulting "mental picture" be called the *internal representation.*

Given that a player "sees" a fairly stable and uniform internal representation, it follows that familiar and meaningful configurations may be recognized in terms of it. The result of visual organization is to classify a tremendous number of possible board situations into a much smaller number of recognizable or familiar board situations. Thus a player can respond to a board position he has never encountered, because it has been mapped into a familiar internal representation.

This report will describe a simulation model for visual organization. It will use transformations which create information corresponding to the perceptual features discussed above, storing them in a computer internal representation.

A heuristic for visual organization

We now examine the problem of modeling the basic visual organization of the GO board. A reasonable goal would be to determine the segmentation of the board, the domains of influence of the stones, and the armies of stones, storing that information in a computer internal representation. Before building the computer model, it is of interest to consider physical processes which give some measure of the influence of physical bodies.

There are many candidates in the physical sciences for the process we desire. For example, white stones could be electrons, and black stones could be protons. Contiguous areas of positive or negative potential could determine the segmentation of the board, and the value of the potential would measure the influence of the stones. Of course, the solution would be discretized to the points of the GO board, and the potential at an unoccupied point could determine how well protected that point is by black or by white.

For another candidate, let the GO board be made of blotter paper and simultaneously place a drop of oil under each black stone and a drop of water under each white stone. Contiguous areas of oil or water would determine the armies and the segmentation of the board. Since the oil and water would spread evenly, the concentration would not indicate the influence of the stones or even their location.

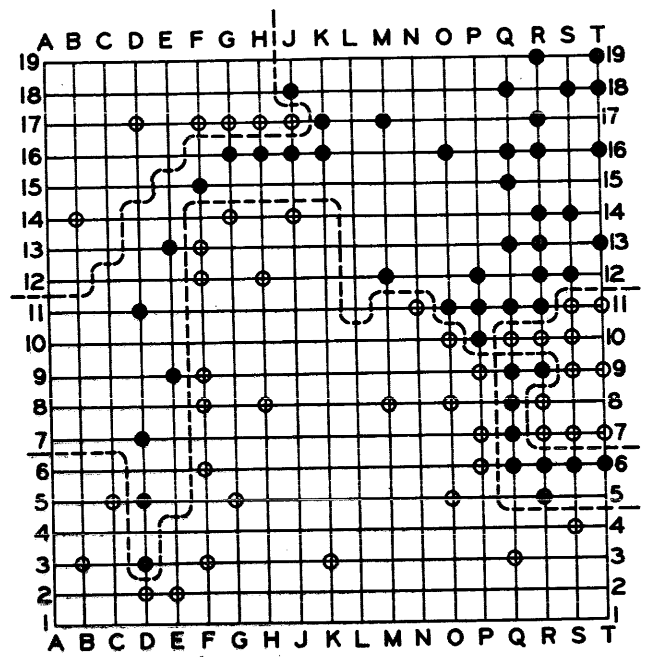
Other possibilities might involve electrical networks or heat conduction, etc. These physical models are considered because they are well defined and easily calculated, whereas the visual process we are attempting to model is ill defined. Let us consider in more detail the first two examples given above. In the center of Figure 1, the electric charge analogy would give the black stone at J10 some weight to the right of the wall of white stones, whereas oil from the black stone would never get past the wall of water spreading from the white stones. Perceptually speaking, the black stone has no influence to the right of the white stones. The oil and water analogy could not differentiate between the two situations in the right hand corners of Figure 1, whereas the electric charge analogy would show four strongly surrounded squares in the lower corner. Thus the oil and water analogy does not reflect our perception of this situation. The finite difference method used by the program was chosen with both of these models in mind, and has the good features of both.

It is assumed that a game is in progress and the board position is stored. The position is transferred to a 19×19 integer matrix by placing 50 for each black stone, -50 for each white stone, and 0 elsewhere. Then each point which is positive sends out a $+1$ to each of its four neighors, and each negative point sends out a -1 to each of its neighbors. These numbers are accumulated as this procedure is repeated four times. Figure 2, which is taken from the game listed at the end of this report, illustrates the results of the visual organization heuristic. The negative integers are indicated by an underline.

Segmentation can be assessed by determining the contiguous areas of positive or negative integers. The dashed lines in Figure 2 indicate the resulting segments. The stones which lie in a segment may be considered to be a group or army. The integer values at a point give a measure of the degree of influence exerted by the stones nearby. The influence of stones of the same color may reinforce one another, whereas the fields from opposing stones seem to repel and cancel one another. Inspection of Figure 2 should convince us that at least a crude resemblance to perceptual processes has been obtained. The array of integers from the visual organization heuristic contains, at least implicitly, information corresponding to an internal representation. This heuristic, together with a routine which is capable of creating an explicit computer internal representation of the resulting information, will be part of a model of visual organization. The specific details of the entire GO-playing program will now be given.

The program

The program is written in ALGOL for the Burroughs B5500 computer. Interaction is provided by remote teletypes. Each move requires 5 to 8 seconds of central



```
 0  2  4  5  6  6  4  1  7  7  6  5  5  5  7 10 59 12 57
 2  4  8 10 10 11 11  2 50 12 10 10  9  9 10 62 16 63 61
 3  7 10 62 10 57 57 56 42 56 13 62 12 11 12 14 63 14 11
 5  8 10  6  0  4 56 57 56 64 12 12 12 62 13 64 64 14 59
 7 10  8  0  7 56  7  6  6  5  8  9  9 11 12 63 15 13 10
 8 62  6  3  6  1 56  8 57  3  3  6  8  8 11 14 64 63 11
 7  9  1  7 54 56 14 13 12  5  4 10  8 10 12 63 65 16 59
 2  0  3 11  6 58 13 62 10  2  7 58  5 12 63 16 65 56  4
 1  4 10 62  6  6 11 10  7  1  2  0 47 49 66 57 50 50 54
 2  5  9 12  7  6 10  9  6  3  2  7 12 48 42 42 50 65 12
 1  4  8 12 54 56 12 11  8  6  8 10 12 14 48 50 42 57 60
 2  5  9 11  5 58 13 62 10  8 10 62 12 62  8 51 49 15 11
 1  3  7 61  4  8 12 10  8  7  8 10 11 13 56 50 50 57 53
 3  3  0  8  3 58 12 10  7  5  6  8 10 13 56 57 58 57 53
 6 11 53 54  1  9 62 10  8  7  7  7 10 62  7  2 58  7  4
 8 12  6  4  1 11 12 10 10 10  8  8  8 10 12  5  6 55  3
 8 61  6 44  5 62 11  9 10 62 10  6  6  8 10 62 11 11  7
 7 11 11 56 63 12  8  6  8 10  8  4  3  4  8 10  9  7  4
 4  6  8  9  9  7  4  3  4  5  4  2  0  2  4  5  5  3  2
```

Figure 2—Results of the visual organization heuristic

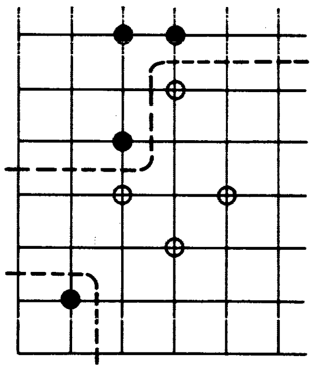
processor time or about 5 to 20 seconds of real time, depending upon the number of users being serviced by the system. The machine code occupies 6300 words of core memory and 5400 more words are required for storage arrays. The program has two distinct parts. Part I has a coordinated set of procedures, including the visual organization heuristic, which operate on the board position to produce a computer internal representation. Part II has a set of procedures which use the internal representation to calculate a move. Part I realizes a model of visual organization for GO, producing an analogue of a human player's perception of the board. Part II is not an attempt at simulation, but a collection of heuristics which may or may not resemble cognitive processes.

Part I

This part of the program consists of a set of computations which transform the board position into a computer internal representation. The internal representation contains an analog of important perceptual features of the board position. This information is stored in seven 19×19 integer arrays in an explicit fashion. That is, the integer values are a direct measure of the features they represent.

For example, consider the features of perceptual grouping and segmentation which have been determined by the visual organization heuristic. It would not be easy to reference this information in the array shown in Figure 2. Another process must create an array which gives a direct measure of the size of the segments and the groups of stones.

Figure 3 illustrates the results of the processes which



603	603	603	603	603	603
603	603	603	704	704	704
603	603	603	704	704	704
704	704	704	704	704	704
704	704	704	704	704	704
501	501	704	704	704	704
501	501	704	704	704	704

Figure 3—Internal representation of grouping and segmentation

calculate perceptual grouping and segmentation. These processes act upon the array of integers produced by the visual organization heuristic to produce an array of integers which are a part of the internal representation. One of these numbers (e.g., 603) may be interpreted as follows: the hundreds indicate the size of the segment which covers that point (600 is a medium-sized segment) and the ones indicate the number of stones which lie in that segment (there are 03 black stones hence 600 + 03 = 603). 500 indicates a small segment with less than 10 empty intersections, and 700 indicates a large segment with more than 25 empty intersections. If no segment covers an intersection, then 0 is stored in the corresponding cell of the array. The empty intersections in a segment are counted as they are a better measure of the safety of the army of stones in a segment. The information is compressed by having the size and the number of stones in the same array for purposes of efficiency only.

The array shown in Figure 3 is typical of the seven arrays which constitute the computer internal representation. Numerical values correspond in an explicit fashion to feature which we have considered to be formed by visual organization.

The second array gives a numerical measure of the influence of the black and white stones. It is an exact copy of the array of integers shown in Figure 2.

The third array measures the number of breathing spaces in a chain. This array is illustrated in Figure 4.

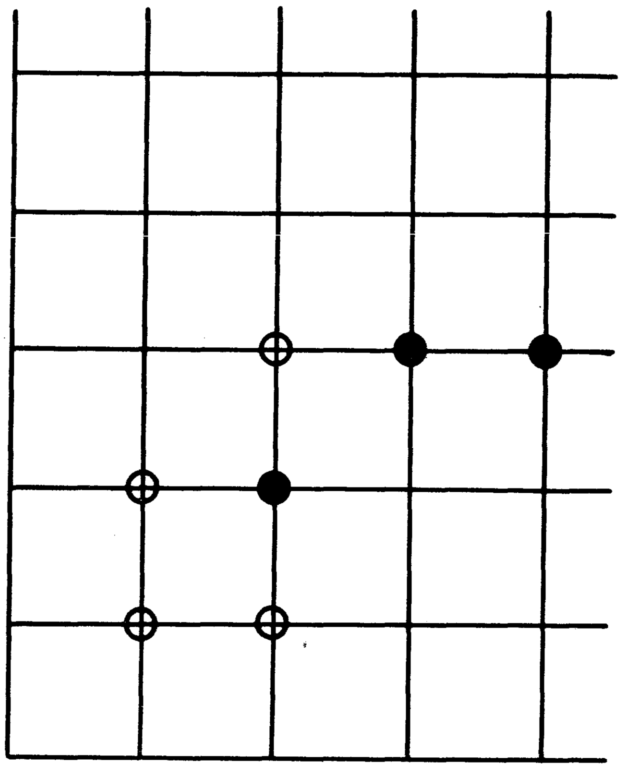
The fourth array measures the number of stones in a chain. This array is calculated in the same fashion as the third array.

The other three arrays of the internal representation contain integers which indicate the color of stones, the color of segments, and the number of stones of each color which are adjacent or diagonal to the points of the board.

Part II

The second part of the program uses the recognition of "familiar" configurations in the computer internal representation as the basis of its calculations. The mechanics of this recognition process will be described first.

Figure 5 illustrates a configuration which the program is able to recognize. At point A, there is a black stone which has only one breathing space left. Point B is an empty intersection. Point C has a black stone which may be part of a safe army of men. Note that the geometric arrangement of this 3-tuple is as important as the three features. The program contains a prototype of this configuration which we shall call a *template*.



0	0	0	0	0
0	0	0	0	0
0	0	2	5	5
0	6	1	0	0
0	6	6	0	0
0	0	0	0	0

Figure 4—Internal representation of breathing spaces

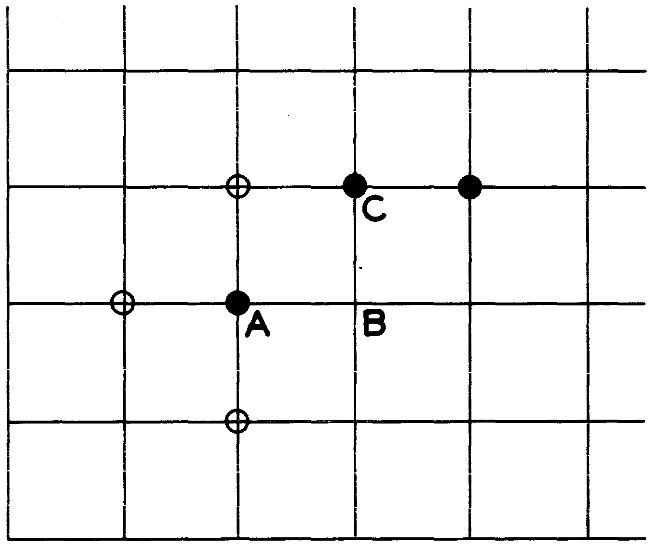


Figure 5—Illustration of a significant configuration

A template consists of an n-tuple of references to the internal representation together with a specification of the geometric arrangment of the elements of the n-tuple. Thus the template for our situation ABC is:

(0,0)	black stone, 1 breathing space
(1,0)	empty intersection
(1,1)	safe black stone.

The pairs of numbers are the relative coordinates of the references. The references themselves must be translated into a numerical form which can be used to process the internal representation, for example:

safe black stone = 601 thru 900 in array 1 and 1 thru 1 in array 7.

That is, a point satisfies the reference "safe black stone" if the value of that point in array 1 of the internal representation lies between 601 and 900 inclusive and the value in array seven equals 1. A 1 in array seven tells us that we have a black stone on that point, and a 601 to 900 tells us that we have at least a medium-sized segment.

The program has the ability to scan such templates to all positions, all rotations, and all reflections of the board, thus recognizing the configuration ABC wherever it occurs. If the template is carefully specified, then configuration ABC can be quite a general occurrence. For example, Point A could be connected to a chain of stones with 1 breathing space left. Allowing this would give the template more generality.

The present program has 85 templates capable of recognizing a wide variety of configurations. A template that matches implies either a move or a heuristic lookahead procedure. In fact, there are two types of templates for these two purposes.

The first type of template specifies a weight of implication and a pair of relative coordinates for the move which is implied. For example, the template for configuration ABC described above would specify

(1,0) 500

which means that a weight of 500 is assigned to the point B in Figure 5. These weights are stored in a special 19 $\times$ 19 array reserved for this purpose. Several templates may imply the same move in which case the weights are summed in this array. The highest sum of weights indicated the best move. Bad moves and illegal moves usually have a negative weight.

One more example of this type of template will be given:

(0,0)	white segment
(1,0)	black segment
(0,0)	weight 40.

This template implies a move with weight 40 at the interface between opposing segments. A weight of 40 is relatively small, hence it will merely give a tendency towards moving in these areas. This template is very general; it can match as many as 100 times in a single scan of the board. It gives a slight tendency to move between opposing armies which helps the program's play.

There are 65 templates which imply moves in the manner just described, the other 20 templates imply a heuristic lookahead. The difference between these two types of templates is that instead of a weight being placed in the weight array, an x is-placed in a 19 $\times$ 19 array as an indication of the template match. The x's are a mark to indicate that a move tree search should be performed in the local area about the x. All of the templates are applied before the search is begun. Figure 6 illustrates a configuration which would be matched by some of the 20 templates, and the location of the x's placed by those templates.

The array which contains the x's is used as a mask to determine the extent of the search and the depth is fixed at two moves for each side. The search is actually performed twice, once for white moving first, and once for black moving first. At the end of these searches, it is noted whether either side can force a capture by moving first. This information tells the program whether a move is necessary to cause or avoid a capture.

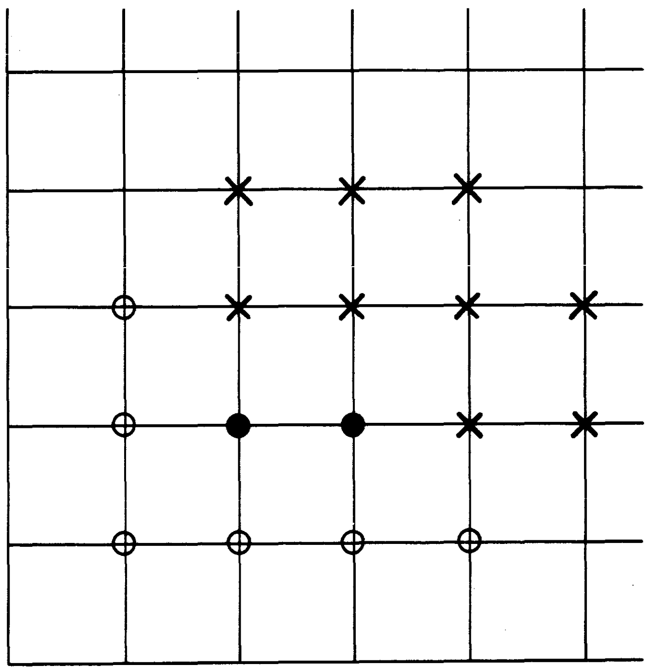


Figure 6—Creation of a mask for lookahead

For example, if black can capture whether he moves first or not, then it is unnecessary for him to move. The decision to move is recorded by placing a weight of 4000 in the array already discussed in connection with the first type of template.

In many cases a depth of two moves is not sufficient to determine whether capture takes place or not. The most common instance of this is known as the "ladder attack" which is illustrated in Figure 7. These situations are characterized by the repeated occurrence of moves which force the opponent to reply or to be captured. In such cases, the search procedure continues to a depth of up to 100 moves to see whether capture finally takes place. No branching takes place during this extension of the look ahead.

When all of the templates have been applied and the heuristic search procedure is through, the program simply chooses the move which corresponds to the highest sum of weights in the array of weights. If the maximum weight is below 100 then the program passes.

This completes the description of the program except for a few minor details of operation. Three seconds are used for the creation of the internal representation and two seconds are used by the template matching procedure. The heuristic search takes from .1 to 4 seconds. A challenger has the option of moving first or second, and can also give the program a handicap of any number of stones on the board. The program is not able to haggle over the final score as GO players often do, hence a referee may be required to supervise the scoring.

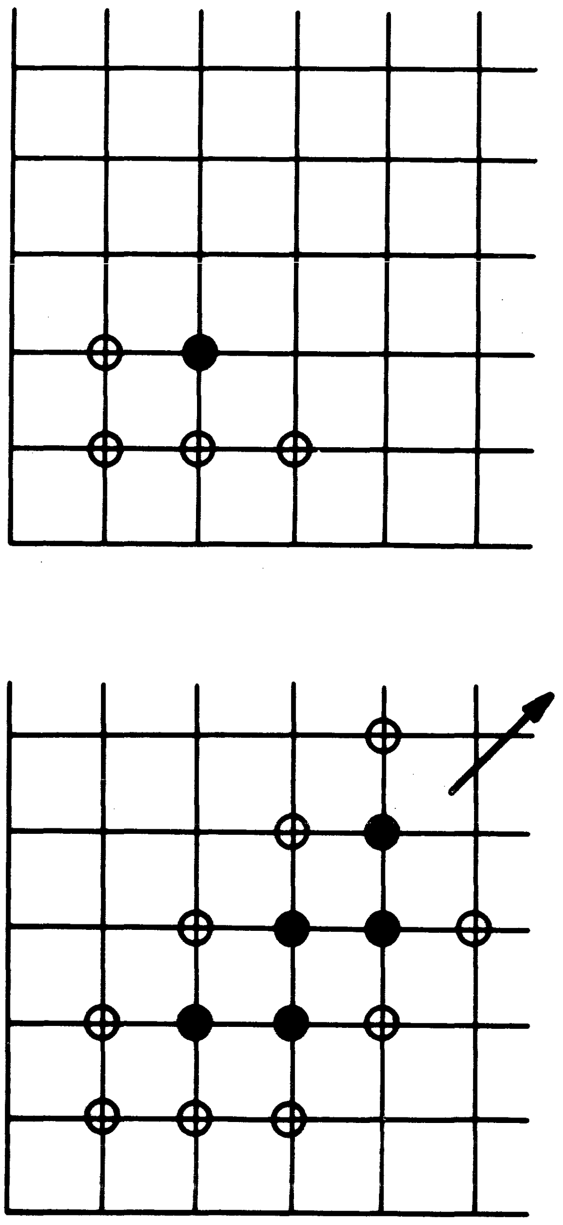


Figure 7—Illustration of the "ladder attack"

RESULTS

The program now has a record of two wins and two losses against human opponents. The opponents can best be described as intelligent adults who know how to play GO, have played from two to twenty games but have not studied the game. The program appears to have reached the bottom rung of the ladder of human GO players. Interested readers are urged to get a GO board and play through the game listed at the end of this report.

The first type of template, those which imply moves, are responsible for about 80 percent of the moves made by the program. These templates give the program fairly good positional play, especially in the first part of the game.

The remaining templates, together with the look-ahead search, are valuable for avoidance of traps which cause the loss of a stone or two. The opponent's play is also restricted since he must play more carefully. The loss of one stone can have a great effect upon the outcome of the game. The program is able to play without these templates, hence without the search, but opponents soon learn to take advantage of this weakness.

The program plays as if it can "see" such things as the influence of stones, the segmentation of the board, and the armies of black and white stones. This alone makes it a reasonable candidate as a model of visual organization for the game of GO. It would be of interest to test the model by performing standard psychological experiments. For example, a drawing of a GO board could be shown to a human subject with the instructions to segment the board with dashed lines. The results could be compared with the segmentation given by the program. Further work may show how important perceptual mechanisms are to the ability of humans to play GO.

APPENDIX: A GAME BETWEEN THE PROGRAM AND MR G. COWAN

The following game, played between the program and Mr. George Cowan, demonstrates that the program has reached at least the bottom rung of the ladder of human GO players. Mr. Cowan is an undergraduate at the University of Wisconsin, and had five games experience at the time of the contest. The even-numbered moves are by the program (white). Moves which resulted in a capture are indicated by asterisks. The comments are by Professor Paul Purdom, a good GO player, who has shown patient interest in the various GO-playing efforts at Wisconsin.

1.	D 3	Q3
3.	E 9	D17
5.	Q16	H12
too early for such a move		
7.	F15	R15
too close to opponent		
9.	O16	M 8
11.	R14	H 8
shouldn't give up the corner		
13.	M12	C 5
15.	D 5	F12
17.	D 7	J17
still important moves remaining in the corner		
19.	M17	R 8
21.	R 5	S 4
23.	D11	K 3
25.	Q11	G 5
27.	Q15	F 6
29.	K16	F17
31.	G16	G17
33.	H16	F 3
35.	E13	E 2
still ignoring upper left side		
37.	Q 7	F 8
wasted move		
39.	J18	S 7
41.	Q 8	O 5
43.	R 9	P 6
45.	S 8	R 7
47.	R 6	S 9
49.	T 8	T 9
51.	S10	T 7*
53.	T10	R10
55.	Q 9	S11
57.	R11	T11*
59.	S12	O 8
61.	S 6	P 7
63.	T 6	P 9
65.	Q 6	Q10
67.	P10	S10
a free gift from black		
69.	O11	Q12
71.	R13	O10
73.	P11	N11
N10 a better move		
75.	R12	H17
77.	J16	D 2
79.	P12	B 3
81.	Q13*	B14
at last! C14 better		
83.	K17	S17
85.	R16	S15
seems to ignore sacrifices		
87.	S14	S16
89.	R17	R18
91.	Q18	F 9
93.	S18	S19
95.	R19*	T17
97.	T19*	T14
both players wasting moves		
99.	T13	F13
101.	T16	T15*

103.	T18	G14
105.	T16*	J14
107.	J10	J11
109.	J12	K12
K11 better		
111.	N10	J13*
wasted		
113.	O 9*	P 8
115.	N 9	L14
117.	M14	L13
M13 better		
119.	K15	K 9
121.	L15	M10
123.	M11	L11
125.	N12*	N13
127.	Q 5	M13
129.	M15	O13
131.	N 7	N 6
should connect N8 and cut stone		
133.	O 7	P 5
135.	P14	Q 4
137.	L10	M 9
139.	L 9	K10
141.	L 8	L 7
143.	K 8	J 8
145.	L12	K 7*
147.	K14	E 5
149.	K13	H10
wasted		
151.	N14	P13
153.	F16	E16
155.	E15	D15
157.	C16	C15
159.	E18	E 7
161.	E17	D18
163.	D16*	C17
165.	B17	C18
167.	B18	F18
169.	E19	H18
171.	K18	G19
173.	J19	F19
the program would be much better if it could recognize eye possibilities		
175.	E16	B19
177.	B16	H15
179.	E12	F11
181.	E14	B 7
183.	B 8	C 8
185.	B 9	R 4
187.	C 9	S 5
189.	A 7	T 5*
191.	B 6	D 6
193.	D 4	D 8
195.	C 7*	C 6
197.	B 7	E 8
199.	D 9	C 4
201.	E10	B 5
203.	E11	E 4
205.	F10	G10
207.	G15	G 9
209.	J15	H14
211.	F14	G12
213.	L 4	L 5
215.	L 3	K 4
217.	K 2	M 3
219.	M 2	N 3
221.	M 4	N 4
223.	L 2	N 2
225.	K 5	J 5
227.	M 1	K 6*
229.	J 1	H 2
231.	J 2	J 3

It was agreed to stop the game at this point. The resulting score was: Mr. Cowan . . . 59, program . . . 66, a seven point victory for the program. Approximately 15 minutes of computer time was used, and the entire contest took less than two hours of real time.

ACKNOWLEDGMENTS

This research was conducted with the support of NIH grant MH 12266 and NSF grant GP 7069.

BIBLIOGRAPHY

1 H REMUS
Simulation of a learning machine for playing GO
Proc IFIP Congress 1962
2 E THORPE W WALDEN
A partial analysis of GO
The Computer Journal Vol 7 No 3 1964
3 I GOOD
The mystery of GO
New Scientist January 21 1965 427
4 O KORSCHELT
The theory and practice of GO
Tuttle Rutland Vt 1966
5 E LASKER
GO and GO-MOKO, the oriental board games
Dover New York 1960
6 A SAMUEL
Some studies of machine learning using the game of checkers
IBM Journal of Research and Development Vol 3 No 3 1959
7 A NEWELL
The chess machine
Proc Western J C C 1955
8 C SHANNON
Programming a digital computer for playing chess

Philosophy Magazine March 1950

9 R GREENBLATT D EASTLAKE III S CROCKER
The Greenblatt chess program
Proc F J C C 1967

10 P GREENE
Networks which realize a model for information representation
Transactions of the University of Illinois Symposium on Self-Organization 1961

11 W KOHLER
Gestalt psychology
Liveright New York 1947

Assembly of computers to command and control a robot *

by LOUIS L. SUTRO

Massachusetts Institute of Technology
Cambridge, Massachusetts

and

WILLIAM L. KILMER

Michigan State University
East Lansing, Michigan

INTRODUCTION

There is a growing consensus among predictors of science that the world is about to witness the evolution of what might be called a new species—the robot. Whereas, animal evolution was a trial-and-error process, robot evolution appears likely to be carefully contrived. Starting where animal evolution left off, that is, with man, robot evolution promises to excel man is some respects, and be excelled by him in others.

To the computer profession, one challenge in this progression is to develop computers for robots that match those that have been found indispensable in men. We are aided in this task by the description of the human nervous system in computer terms by physiologists such as Warren McCulloch.

With his description before us, we have devised working models of two of the five principal computational domains which he identifies in the nervous system of vertebrates, including man. Others are devising working models of other domains. Implemented in light, portable hardware and connected together, these computers promise to provide intelligence for a system that will sense its environment, move about and perform useful tasks.

Who needs a robot? Everyone who would like help with tiring chores. However, early models with large arms and wide wheelbases cannot move around the home or office. One need that has led to the development about to be described is exploration of the planet Mars. For this task, robot development is being pursued not as an end in itself but as a framework within which to develop an automatic visual subsystem. A second need is for a computer to command a system receiving several forms of input, such as sight, sound, touch, and reports on its own movements. Here again robot development provides the framework for the computer development.

As well as can be determined,[1] the surface of Mars is open country where a wide-wheelbase vehicle should be at home. More to the point, the only exploration there for a decade or more will have to be either by a remote-controlled or an automatic vehicle. The distance is such that a single bit of information requires 15 minutes, on the average, for transmission from Mars to earth. With such a transmission delay, remote control seems hardly practical. An automatic vehicle or robot thus seems imperative.

While the surface of Mars is colder than the surface of the earth, there may be hot spots due to volcanic or other sub-surface activity. All the moisture on Mars, according to our instruments, is in the form of either gas or ice. The atmospheric pressure is too low to hold it as water, but it might pass through the water phase in these hot spots, lasting as water long enough to make possible life as we know it.[2]

To go to these hot spots, if indeed they exist, poke

* The work reported here was supported in part by NASA Office of Space Sciences and Applications, Bioscience Programs, under Contract NSR 22-009-138, in part by NASA Electronic Research Center under Grant NGR 22-009-140, in part by Air Force Office of Scientific Research under Grant AF-AFOSR-1023-66, and in part by the U. S. Air Force, Wright-Patterson Base, through contract AF 33(615)-3885.

around them, pick up and examine samples seems the best way of finding out what is there. Even if there is no life on Mars, there are cliffs formed at the edges of craters, that need to be examined for their geology. The craters need to be climbed into and out of. To go from one crater to another, crossing must be made of the ravines called "canals".

Research and development

The robot design described here began as an effort to design eyes for the artificial intelligence that Marvin Minsky and John McCarthy called our attention to, in the fall of 1958. Persuaded that eyes for artificial intelligence could be achieved only by employing ideas from animal vision, one of us (Sutro) approached Dr. McCulloch for advice. The collaboration that ensued led first to an analytical model of an animal retina that recognizes objects, namely, the retina of a frog.[3,4] It led next to a proposal to NASA to develop means of reducing, for transmission to earth, pictorial data acquired in the search for evidence of both life and geological changes on Mars. Supported then by the NASA Bioscience Programs, we undertook this in the manner Dr. McCulloch and we thought best, namely, to model animal vision in lightweight, low-power hardware. Study of frog vision showed how recognition of a simple shape (a bug) can be achieved in two levels of computation, but it did not carry far enough the data reduction we felt was required. Needed, we felt, was reduction of a stereo pair of images on Mars to a pair of line drawings with shading, as we primates do. Geologists and biologists make line drawings with shading to represent what they see. The lines portray edges, angles and silhouettes. The shading conveys the brightness of surfaces.

Man forms in his head a model of what he observes. Formation of a line drawing with shadings is a stage in the computation of this model. However, as Dr. McCulloch points out, the vision of a primate cannot be modeled by itself. Data flows not only inward from the images, but outward from the brain to adjust the filters, focus, convergence and direction of gaze that select what will flow inward. For a visual system employed in a single position on Mars, these adjustments can be either preset or changed by commands from earth, but when the system is required to move about, the commands to adjust it can scarcely be sent from earth. They have to be generated on site.

To develop a command computer one of us (Kilmer) undertook to model the part of the vertebrate brain that decides from information received through all the senses what class of thing the animal will do from moment to moment. This is the core of the animal's reticular formation, extending through its brain stem and the length of its spinal cord. Support for its development came first and continues from the Air Force Office of Scientific Research, came then from NASA's Electronic Research Center, and comes now from the U.S. Air Force bionics programs.

Cameras and computers under development are pictured in Figure 1. At the left is a binocular pair of TV cameras of which sufficient study has been made to indicate that each camera can be built to view the world through both wide-and narrow-angle lenses. Receiving the output of the camera is the visual first stage computer which enhances contrast in an image, as an animal retina does. Next to it are switching filtering and comparison structures, we call the visual second stage computers. A model of the environment consists of relations formed in this second-stage visual computer and stored in the visual part of the relational computer. A line, which indicates sharp change in luminance, is a relation of high spatial frequencies. Shading, which indicates the difference in luminance of areas, is a relation of low spatial frequencies. Each filter passes one band of frequencies more than others. Commands to adjust filters, focus and direction of gaze are shown as arrows rising from the command computer in Figure 1. Since these commands will pass through structures not shown, the arrows are not drawn directly to the cameras and visual computers.

Note the dashed boxes. The present locator of edges and shades, represented by a solid box, forms a stereo pair of monocular line drawings. The dashed box marked "binocular" represents computation now operating separately to determine that pairs of points in the left and right views are stereoscopic (stereo), that is, representative of the same point in three-dimensional space. Binocular, or range-finding, computation will be merged with the locator of edges and shades.

At first, we called a vehicle designed to carry this system "rover". As we came to conceive of it with other senses, beside vision, and other effectors, beside wheels, we renamed it "robot."

Biological computers

From his life-long study of the human nervous system,[5] Dr. Warren McCulloch has concluded that the essential features of its computations provide a good basis for the design of a robot. Although as a neurologist, psychologist and physiologist, he is aware of the difficulties involved in embodying mental functions in physical devices, he has nevertheless developed a simplified model of a vertebrate brain.

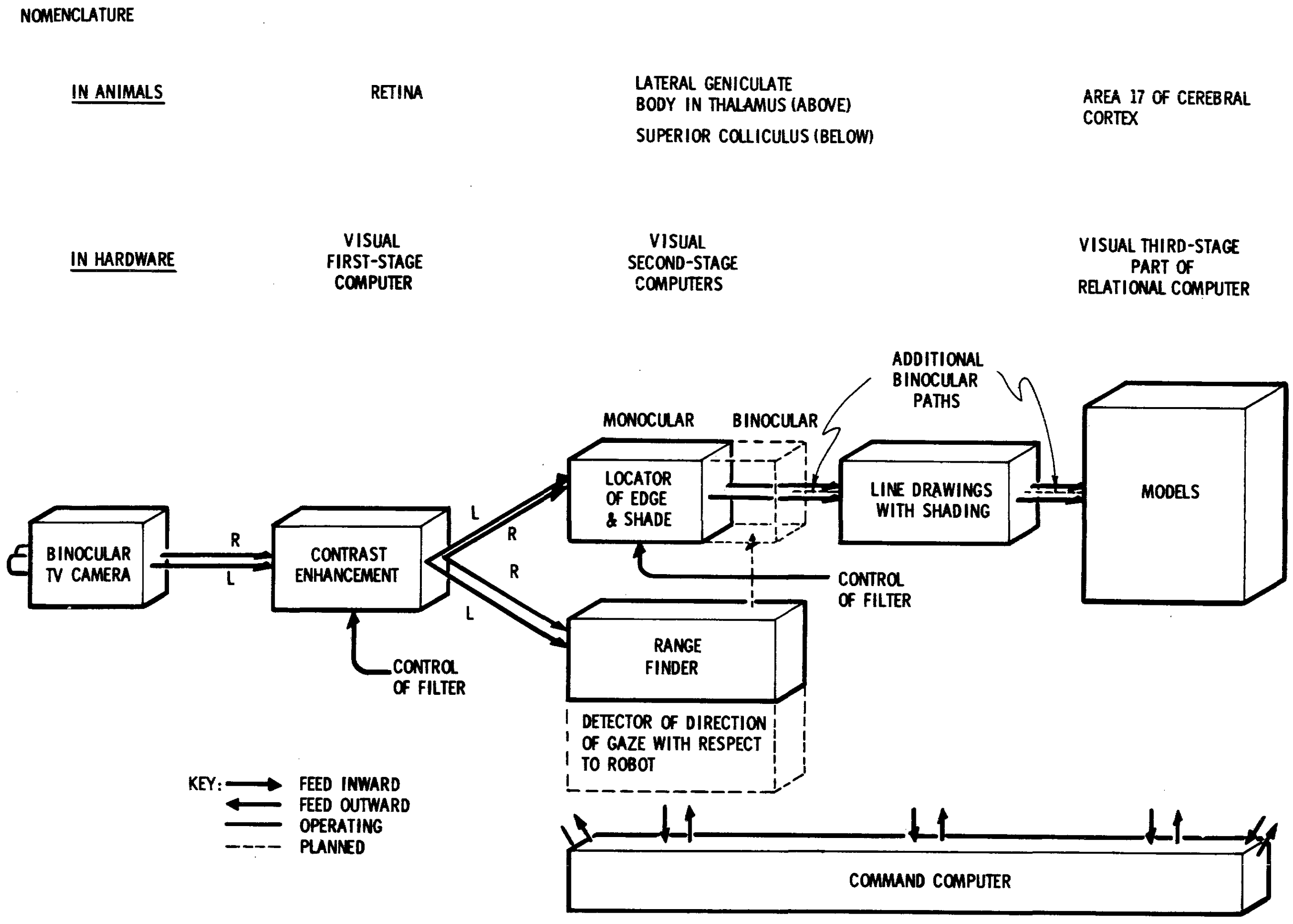

NOMENCLATURE
IN ANIMALS
RETINA
LATERAL GENICULATE BODY IN THALAMUS (ABOVE)
SUPERIOR COLLICULUS (BELOW)
AREA 17 OF CEREBRAL CORTEX
IN HARDWARE
VISUAL FIRST-STAGE COMPUTER
VISUAL SECOND-STAGE COMPUTERS
VISUAL THIRD-STAGE PART OF RELATIONAL COMPUTER
ADDITIONAL BINOCULAR PATHS
MONOCULAR
BINOCULAR
BINOCULAR TV CAMERA
R
L
CONTRAST ENHANCEMENT
CONTROL OF FILTER
LOCATOR OF EDGE & SHADE
LINE DRAWINGS WITH SHADING
MODELS
CONTROL OF FILTER
RANGE FINDER
DETECTOR OF DIRECTION OF GAZE WITH RESPECT TO ROBOT
KEY: FEED INWARD
FEED OUTWARD
OPERATING
PLANNED
COMMAND COMPUTER

Figure 1—Computers being developed. Feed outward for perception is indicated in the control of filters

His intention is merely to suggest an organizational structure necessary for efficient robot performance.

Figure 2 outlines his model of the vertebrate nervous system, identifying what he feels are five principal computational domains and their chief functional connections. At the left is the *retina*, consisting of three layers of cells, two of which seem to perform most of the computation. The eye is shown as representative of the senses because its computational capacity qualifies it as a principal computer; it is the foremost data source to the primate brain, providing two million of its three million inputs. Other senses shown are acoustic (represented by the cochlea), vestibular and somatic.

At the upper left is the *cerebrum*, which Dr. McCulloch calls the "great computer" and in which computation is carried out in many layers. Each of these, if unfolded from our brains, would be about the size of a large newspaper.

The computer which controls all others is shown at the center right. It is the *reticular core* of the central nervous system which extends from the base of the cerebrum through the spinal cord. It directs the main focus of attention so as to determine what type of activity is to occur from moment to moment. By committing the animal to one or another mode of behaviour, it controls all other computers and, through them, the whole organism.

Clusters of nerve cells at the base of the cerebrum comprise the *basal ganglia*, a computer shown at the lower left of the figure. Here are programmed all of the innate or learned total action patterns of the body, such as feeding, walking or throwing a ball. Additional programs are acquired through the growth of connections to the motor-control nerve cells, shown along the bottom of the illustration.

Completing the list of principal computational areas is the *cerebellum*, shown at the top of Figure 2. It computes the termination of a movement, such as reaching to touch an object, and requires inputs from the vestibular system, to detect tilt and acceleration of the head, and from skin- and muscle-sense cells to detect posture and the nature and position of what is being touched.

Interconnected with the principal computers are switching structures, such as the thalamus, colliculus, and cerebellar anteroom. In fish, amphibians, and birds the superior colliculus perceives form and movement; in visual mammals, it determines the direction of gaze and reports by thalamic relay the cues of seen

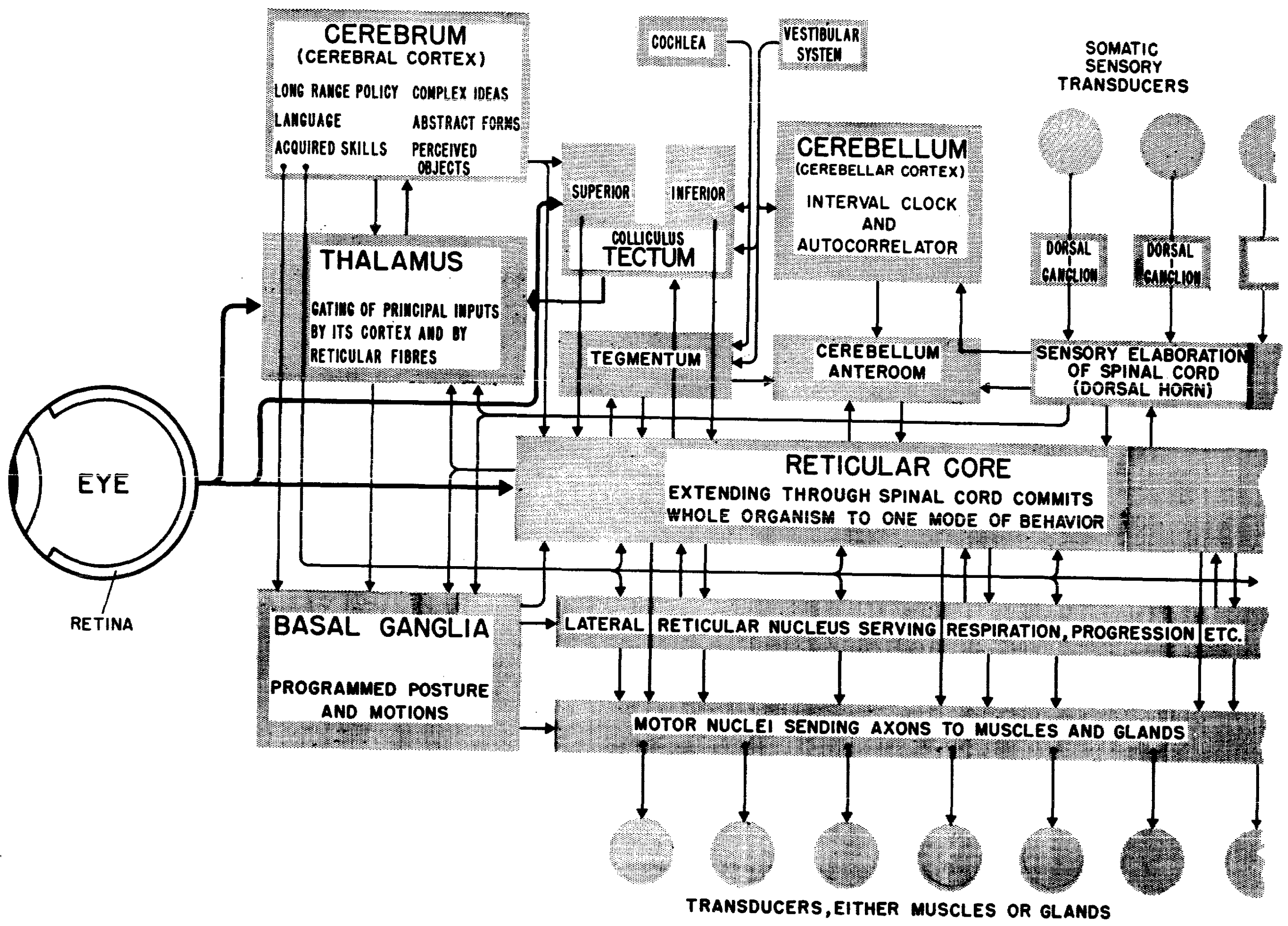

CEREBRUM
(CEREBRAL CORTEX)
LONG RANGE POLICY
COMPLEX IDEAS
LANGUAGE
ABSTRACT FORMS
ACQUIRED SKILLS
PERCEIVED OBJECTS
COCHLEA
VESTIBULAR SYSTEM
SOMATIC SENSORY TRANSDUCERS
CEREBELLUM
(CEREBELLAR CORTEX)
INTERVAL CLOCK AND AUTOCORRELATOR
SUPERIOR
INFERIOR
COLLICULUS
TECTUM
THALAMUS
GATING OF PRINCIPAL INPUTS BY ITS CORTEX AND BY RETICULAR FIBRES
DORSAL GANGLION
DORSAL GANGLION
TEGMENTUM
CEREBELLUM ANTEROOM
SENSORY ELABORATION OF SPINAL CORD (DORSAL HORN)
EYE
RETINA
RETICULAR CORE
EXTENDING THROUGH SPINAL CORD COMMITS WHOLE ORGANISM TO ONE MODE OF BEHAVIOR
BASAL GANGLIA
PROGRAMMED POSTURE AND MOTIONS
LATERAL
RETICULAR NUCLEUS SERVING
RESPIRATION, PROGRESSION
ETC.
MOTOR NUCLEI SENDING AXONS TO MUSCLES AND GLANDS
TRANSDUCERS, EITHER MUSCLES OR GLANDS

Figure 2—Block diagram of generalized vertebrate nervous system. Feed-outward paths are not shown

motion to the secondary visual cortex. The inferior colliculus is concerned with auditory and vestibular inputs as well as with orientation of the body image in space. Below the colliculus is the tegmentum, which is concerned with the relations between things seen, heard, and felt and the control of progression and postural righting actions.[6]

Around the reticular core are specialized structures that could also be called computers, such as the nucleus of nerve cells that control respiration and other routine bodily functions, and the dorsal horn of the spinal cord, through which pass inputs from sensory cells. Note that the reticular core acts on all other computers and that they report to it. It reaches decisions with the aid of raw data from the sensory systems but its main input comes from the other computers.

The computers of Figure 2 are shown as they are arranged in animals with horizontal spines. Monkeys and man have the same computers in approximately the same relation, but the arrangement is vertically distorted, with the cerebrum, now very much larger, at the top.

All these computers have a common ancestry. All evolved from the central computer, the reticular core, and in so doing have established only those interconnections necessary for efficient communication with it. Out of the reticular core has thus evolved the complexity necessary to meet the demands of the entire system.

An engineering analog

Figure 3 is a diagram analogous to Figure 2, labelled with engineering terms to suggest how the animal system can be simulated. For example, in place of the retinas are the cameras and the visual first-stage computer, previously shown in Figure 1. First stage computers receive imputs from all of the senses—auditory vestibular and somatic sensory. Each is called a computer rather than a precomputer or preprocessor to indicate that it recieves feed-outward signals from the central computers.

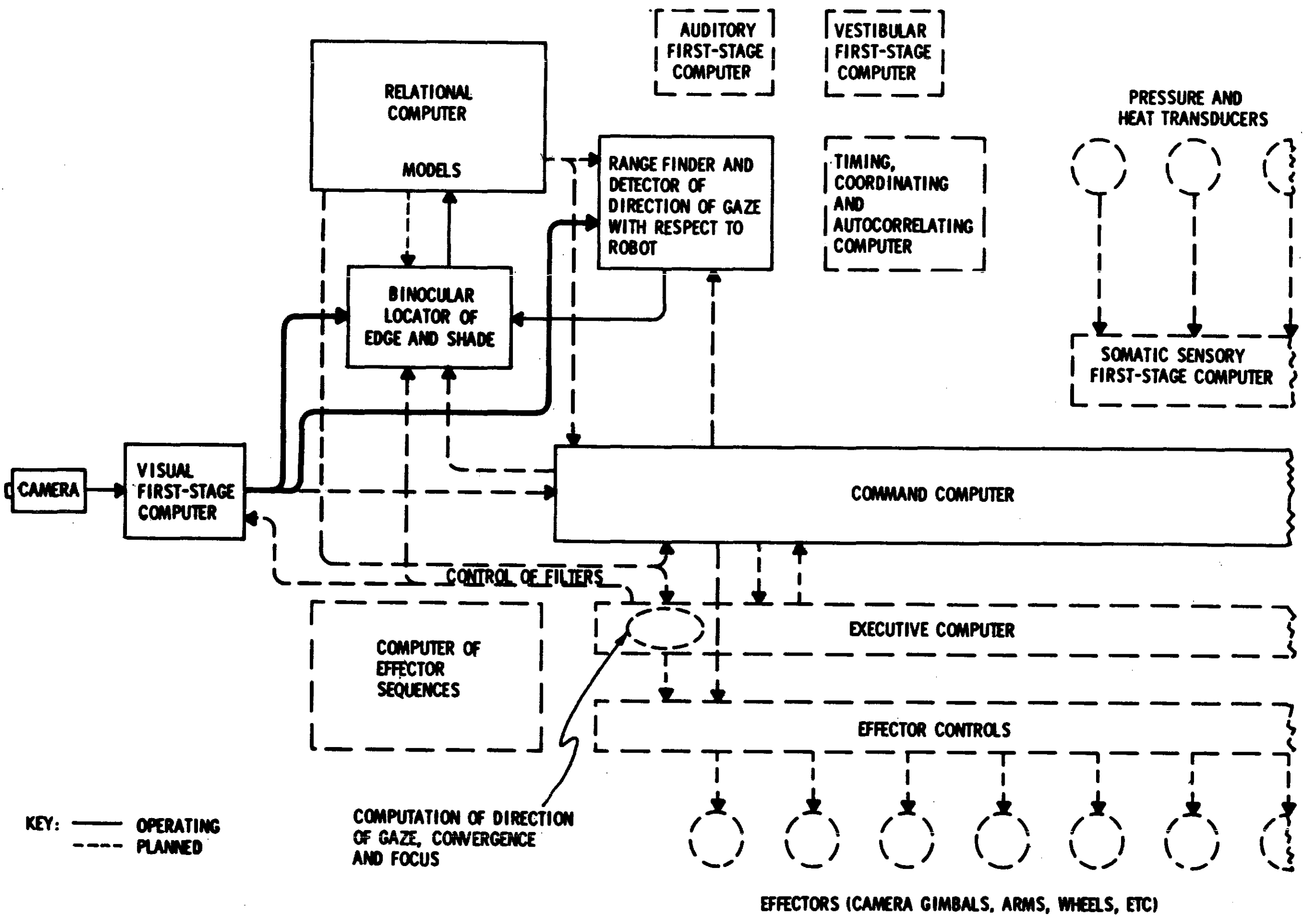

AUDITORY FIRST-STAGE COMPUTER
VESTIBULAR FIRST-STAGE COMPUTER
RELATIONAL COMPUTER
MODELS
RANGE FINDER AND DETECTOR OF DIRECTION OF GAZE WITH RESPECT TO ROBOT
TIMING, COORDINATING AND AUTOCORRELATING COMPUTER
PRESSURE AND HEAT TRANSDUCERS
BINOCULAR LOCATOR OF EDGE AND SHADE
SOMATIC SENSORY FIRST-STAGE COMPUTER
CAMERA
VISUAL FIRST-STAGE COMPUTER
COMMAND COMPUTER
CONTROL OF FILTERS
COMPUTER OF EFFECTOR SEQUENCES
EXECUTIVE COMPUTER
EFFECTOR CONTROLS
KEY: —— OPERATING
---- PLANNED
COMPUTATION OF DIRECTION OF GAZE, CONVERGENCE AND FOCUS
EFFECTORS (CAMERA GIMBALS, ARMS, WHEELS, ETC)

Figure 3—Engineering analog of generalized vertebrate nervous system

Other substitutions are as follows:

command computer	for	reticular core
relational computer	for	cerebral cortex
timing, coordinating and autocorrelating computer	for	cerebellum
computer of effector sequences	for	basal ganglia (nucleii)
executive computer	for	lateral reticular nuclei

The connections to the command computer shown in Figure 3 are only those referred to in this paper. Eventually this computer will connect to every subsystem and every subsystem will connect to it. Examples of sensory subsystems are visual, auditory, vestibular, contact and kinesthetic. Examples of effector subsystems are vehicle, arms, camera focus and camera gimbals.

When the feed-outward paths are added, and control loops are drawn through the environment in the manner spoken of in an earlier section, the system is seen to be composed entirely of closed loops.

Logic in biological and electronic computers

On the one hand, we have the nets of the nervous system; on the other, the contrived logic of electronic computers. In a living nerve net, branches interconnect; information from every source mixes at many places with information from every other source, and affects every output. This is called an "anastomotic net".

The electronic computers we are designing at present are not programmable general purpose (GP) machines. A GP computer is primarily intended for sequential computation on stored data. It is adept at taking data from one part of memory, modifying it, then putting it back into memory. The need here, however, is to compute on a large-volume stream of data entering from the outside. Accordingly, special purpose (SP) computers are being designed in which computation is performed on the data soon after it enters the system from one or many sources. There is no more than buffer storage between the entrance and the computation.

For each of the "five principal computational domains" described in a previous section, we aim to build an electronic approximation to an anastomotic net. To do this we need to:

1. Approximate its functions;
2. simulate these approximate functions in a configuration that is realizable in hardware, and
3. realize these approximate functions in an SP computer.

For example, to design a model of a retina we first approximated its function of enhancing contrast by the function described in a later section. We call this function a "visual first stage computer", are simulating it in a GP computer, and have partially constructed an SP computer to do it. Other functions of the retina will also be simulated such as enhancement of color contrast.

To design a command computer we first approximated the functions of the reticular formation in animals, described in a later section. We call the successive simulations S-RETIC and STC-RETIC. Design of an SP computer to perform these functions is under way.

Memory

What characterizes the evolution of S-RETIC into STC-RETIC is the addition of storage or memory. The same will happen in the evolution of the visual computers. They include no memory now because, for early Mars landings, the intent is only to reduce pictorial data on Mars for reconstruction and viewing on earth. The command computer, on the other hand, can be conditioned to respond to a pattern of stimuli, can drop out this conditioning in the process called habituation, yet can pick it up agian. The input-output relations that constitute this memory are stored in tables in the simulation of the command computer, but will be stored in adaptive elements in the hardware design. The relations are between different forms of stimulus and response. For example, STC-RETIC, when reinforced, remembers the mode of behavior it selected in response to a pattern of inputs. The relational memory of Figures 1 and 3 will remember fine details of this coarse relation. One reason for thus extending the organization of STC-RETIC is that animals have successfully done this. The cerebral cortex evolved from the core of the reticular formation.

An object will be stored, not as a picture, but as stimuli which cause the robot to do something: run from the object, pick it up, experiment with it. A model, which is a stored response to an object, can either be built in or learned. If we construct a robot, it will be to perform a useful task, not to show us what is in its head. Ability to draw pictures is a skill for which the aptitude can be built in and proficiency learned. However, in a device that is only part of a robot, such as the camera-computer chain proposed for an early Mars landing, the only useful output is a stereo pair of line drawings with shading. In the evolution from the present visual system to the visual subsystem of a robot, the segments of what is now a line drawing are likely to become features of surfaces in three-space which can be formed into drawings only by an added process.

We would call this memory an "assocaitive" computer were it not that this term has a different meaning in engineering than in physiology. In engineering, it means "content addressable", which is not an adequate memory from out point of view. As Dr. McCulloch puts its, "The memory we need should be addressed on the basis of relations, appropriate to its mode of behavior. We know *a priori* that spatial relations, constituting objects, form categories both to guide locomotion, etc., and to form the bases of descriptions. Size and precise shape are secondary. Just as a baby has a built-in mechanism to find and follow faces, and only later to recognize particular ones, so our robot should see abstractions first and qualify them later in terms of corners, angles, surfaces and edges, as we do a face, in terms of eyes, nose, mouth and eventually ears. Since a relation can be described in a sentence, a computer, designed with relational addressing for visual relations, can be extended to verbal ones".

First stage of visual computation

The scene before an animal eye or a television camera can be described as a mosaic of luminances. If you doubt this, take a luminance meter, such as a photographic exposure meter, and aim it in a sequence of directions from left to right along a horizontal line; then in the same sequence of directions along a second horizontal line, below the first; then along a third and a fourth horizontal line; and so on until you have scanned a square pattern or raster. The readings of the meter are the mosaic of luminances. Inverted and exchanged left for right, this same mosaic is the image at the back of your eye or on the face of a television camera tube.

As the luminance changes from point to point across the image, there is a luminance gradient which can be detected and represented by a dot. Sufficient dots form a line and sufficient lines a line drawing.

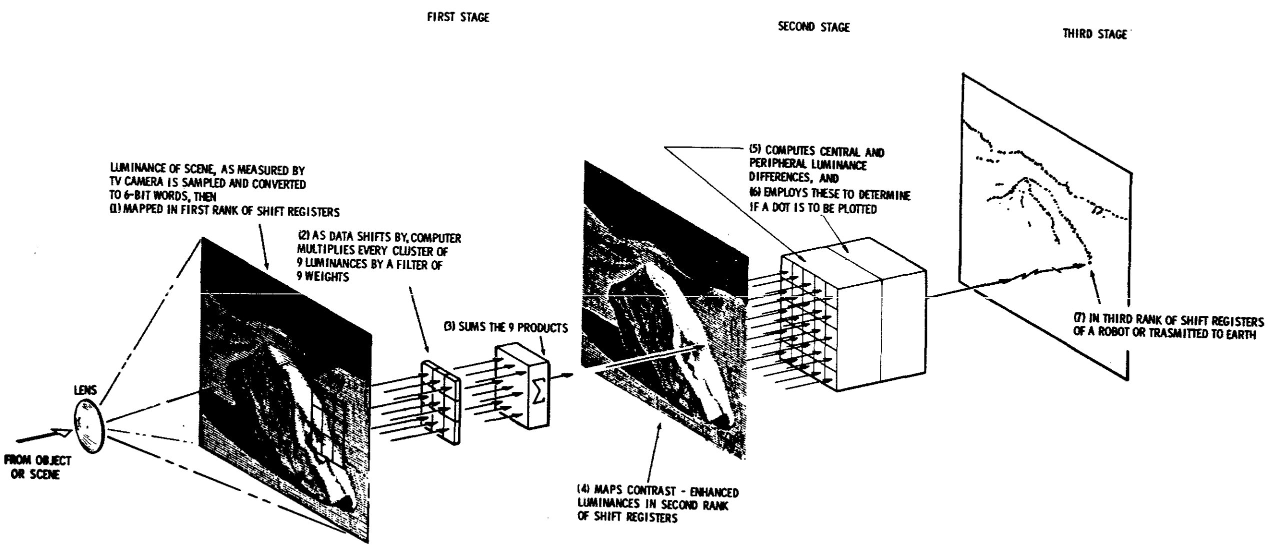

FIRST STAGE
SECOND STAGE
THIRD STAGE
LUMINANCE OF SCENE, AS MEASURED BY TV CAMERA IS SAMPLED AND CONVERTED TO 6-BIT WORDS, THEN (1) MAPPED IN FIRST RANK OF SHIFT REGISTERS
(2) AS DATA SHIFTS BY, COMPUTER MULTIPLIES EVERY CLUSTER OF 9 LUMINANCES BY A FILTER OF 9 WEIGHTS
(3) SUMS THE 9 PRODUCTS
LENS
FROM OBJECT OR SCENE
(4) MAPS CONTRAST - ENHANCED LUMINANCES IN SECOND RANK OF SHIFT REGISTERS
(5) COMPUTES CENTRAL AND PERIPHERAL LUMINANCE DIFFERENCES, AND
(6) EMPLOYS THESE TO DETERMINE IF A DOT IS TO BE PLOTTED
(7) IN THIRD RANK OF SHIFT REGISTERS OF A ROBOT OR TRASMITTED TO EARTH

Figure 4—Levels of visual computation performed on a mosaic of luminances. At levels 1 and 4, horizontal lines in the image represent bands of luminances; thus, the squares drawn on the image are oversize. The images are neither inverted nor turnedright for left as they should be

Addition of low resolution (low spatial frequency) changes in luminance gives the drawing shading.

Whether we take animal vision as our model, as we are doing here, or develop designs independent of the animal, as others do, we find that three stages of computation are needed to achieve the abstraction which we call a "line drawing with shading" and make it useful in the command and control of a robot. As shown in Figure 4, the first stage enhances contrast. The second stage forms line drawings which are either mapped in the third stage or, as proposed for an early Mars landing, transmitted to earth. A part of the second stage not yet tied into the sequence of Figure 4, determines the range of dots mapped in the third stage. Still another part, to determine shading in the line drawing, has been simulated by an artist and will be automated and tied in later.

The stages presently operating as a sequence are broken down into levels in Figure 4. Continuous luminance measurements made by the TV camera are sampled, converted to 6-bit digital words, and, in level 1, mapped. At level 2, parallel computation is performed on a number of luminance measurements which, for illustration purposes, is shown as 3 $\times$ 3, although in present experiments, it is 13 $\times$ 13.

The Jet Propulsion Laboratory (JPL) of the California Institute of Technology has improved, by computer, the quality of pictures sent back from the moon and Mars and x-ray radiographs of medical cases. Their objective is "to make selected features easier to see. This might require suppression of useless data such as random noise and background shading or perhaps amplification of fine detail."[7]

Our first objective on the other hand, is to reduce pictorial data for both transmission from Mars to earth and for reconstruction there. Only after it has gone through reduction and transmission do we want to make it easier to see. Our second objective is to reduce pictorial data to enable a robot to see. Yet our objectives and JPL's appear achievable in the same way, namely, by operations on the spatial frequencies in the image.

The output of a TV camera is a waveform and as such is analyzable into frequencies of luminance amplitude in the horizontal direction of sweep of the camera beam. Since the TV raster is made of many lines, measured vertically, the image on the face of the camera tube is also analyzable in the vertical direction. The frequencies of luminance amplitude in all possible directions within the plane of the image are called "spatial frequencies."

Our equipment operates on these frequencies and amplitudes by employing digital filters, although the filters can be "analog" in the sense that computer designers use this term. A digital filter is a matrix of weights which can be made to operate on a matrix of luminances* by either of two methods. One is convolution of the filter with a sub-matrix of luminances, as in equations 1 and 2 below. The other

* "Luminance" from now on is used to represent luminance measurement.

is convolution of the filter with the entire matrix of luminances that is the image.

To amplify fine detail, a filter should pass high spatial frequencies, reject low and do this in all possible directions in the plane of the image. Equation 1 shows a simple filter designed for this purpose, convolved with a sub-matrix of uniform luminances.

$$\underset{\text{Luminances}}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}} * \underset{\text{Filter } G_1}{\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}} = 0 \qquad (1)$$

Since the convolution sum is zero, adding it to the central luminance in the sub-matrix produces no effect. That is, there is no fine detail in the image to be amplified. Given a sub-matrix of different luminances and the same filter, the convolution produces:

$$\underset{\text{Luminances}}{\begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \\ 1 & 2 & 2 \end{bmatrix}} * \underset{\text{Filter } G_1}{\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}} = 3 \qquad (2)$$

Added to the central luminance, it enhances contrast between this central luminance and the 1 at its left.

If the band of 1's in the above matrix of luminance, extends indefinitely up and down and to the left, the band of 2's extends indefinitely up and down and to the right, the filter may be convolved with all possible 3 × 3 matrices to form the new matrix:

$$\begin{bmatrix} & & & \vdots & \vdots & \vdots & & & \\ \cdot & \cdot & 0 & -3 & 3 & 0 & 0 & \cdot & \cdot \\ \cdot & \cdot & 0 & -3 & 3 & 0 & 0 & \cdot & \cdot \\ \cdot & \cdot & 0 & -3 & 3 & 0 & 0 & \cdot & \cdot \\ & & & \vdots & \vdots & \vdots & & & \end{bmatrix}$$

Adding this to the original matrix produces:

$$\begin{bmatrix} & & & \vdots & \vdots & \vdots & & & \\ \cdot & 1 & 1 & -2 & 5 & 2 & 2 & \cdot & \cdot \\ \cdot & 1 & 1 & -2 & 5 & 2 & 2 & \cdot & \cdot \\ \cdot & 1 & 1 & -2 & 5 & 2 & 2 & \cdot & \cdot \\ & & & \vdots & \vdots & \vdots & & & \end{bmatrix}$$

Thus where contrast exists, the high luminance has been made higher, the low lower. Both of the above methods of amplification or enhancement are called "lateral inhibition" because, while the center of the matrix is excitatory, the periphery is inhibitory.[8] The bands of enhanced light and dark that result from application of the filter are called Mach bands after Ernst Mach who first described them.[9] Our retinas perform this operation so that we see Mach bands wherever there is a wide step in luminance.

The multiplications in the above operations are pictured in Figure 4 as taking place in level 2, the summation in level 3.

To avoid adding the convolution sum to the central of the original luminances we can double the central weight in the filter. Furthermore, to keep most convolution sums in scale we can multiply the filter by 1/8. To satisfy these conditions, we select weights such that the central one equals twice the absolute value of the surround and the sum is one, e.g.,

$$G_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 16 & -1 \\ -1 & -1 & -1 \end{bmatrix} \text{x } 1/8$$

$$= \begin{bmatrix} -1/8 & -1/8 & -1/8 \\ -1/8 & 2 & -1/8 \\ -1/8 & -1/8 & -1/8 \end{bmatrix} = 1 \quad (3)$$

A way of keeping this on scale when applied to spots and edges of maximum contrast is given in Appendix A.

Second stage of visual computation

The second stage is being designed not only to reduce the contrast-enhanced image of a scene to a line drawing with shading, but to permit perception and recognition. Here the filter can be adjusted, as the image moves past it, to seek a match with features in a stored model. However, in each experiment so far performed, the filter has been held constant.

Figures 5 and 6** illustrate the formation of a line drawing of the scene in the background of Figure 7. The scene is lighted from high on the left, casting a short shadow at the right of the rock. Note that the contrast of the rock against its background is weak in the digitized raw data of level 1 (Figure 5, left), but strong in the contrast-enhanced data of level 4 (Figure 5, right). If there is contrast between adjoining luminances, the filter will make the dark side of an edge darker and the light side lighter, thus forming Mach bands along the edges of the stone, the stick, the hills and the crater.

The vertical lines in the displays of Figure 5 are due to the method by which the computer of Figure 7 acquired pictorial data. As the electron beam of the camera scans each horizontal line of the raster, the computer commands the reading of one luminance

** A stereoscope for viewing this illustration may be obtained from Air Photo Supply, Yonkers, N.Y.

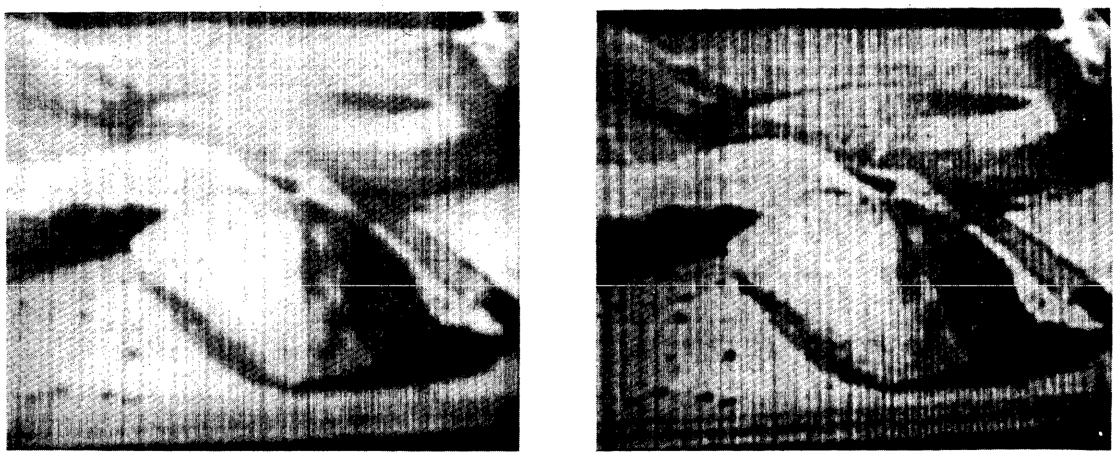


Figure 5—Left-eye-view scope displays at level 1 (left) and level 4 (right) in the computation of the line drawings of Figure 6

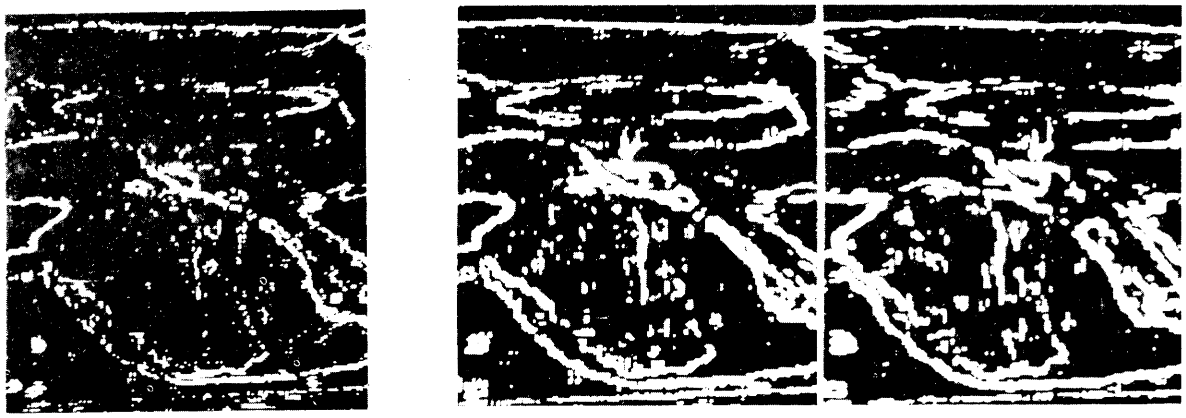
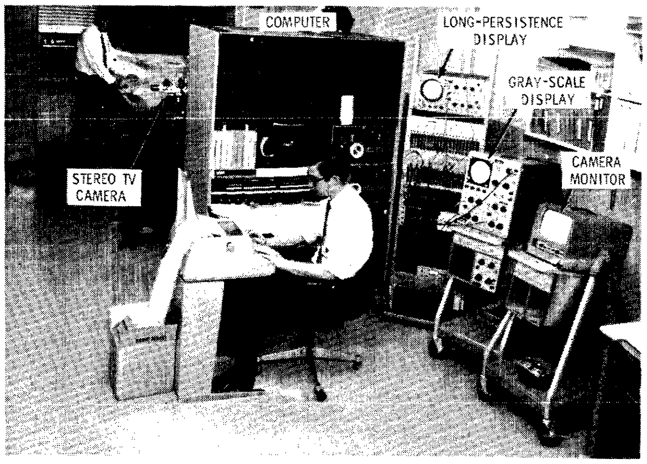


Figure 6—Line drawings formed at level 7: Fine-line left-eye-view (left), coarse-line stereo pair (right). To see in stereo, look through a stereoscope at the crater, glancing nearer occasionally at the edges of the rocks and hills

measurement.[10] Since the computer employs the same command as long as it can, successive measurements are on vertical lines. The unevenness in the spacing of the lines is due to nonlinearity in decoders of the display.

Figure 6 (left) is the result of applying, at levels 5 and 6, an edge-detecting filter and threshold, that represent steep changes (gradient) in luminance by a fine line. Figure 6 (right) is the result of applying a filter and threshold that represent such changes by a wider and, in this case, more continuous, line. It is thus possible to pick different features from a scene by employing at level 5, different edge-detecting filters and, at level 6, different thresholds. The filters employed at level 5, in producing these illustrations, detected the horizontality and verticality of edges separately, but this information was not kept separate in making the line drawings. For details see Appendix A.

When contrast is high in the scene, as in Figure 8 (left), first stage visual computation can be skipped. Figure 8 (left) is a photograph of the TV monitor (not the oscilloscope display as in Figure 5, left) when the scene was lighted from the back left, creating higher contrast than was the case for Figure 5 (left). Levels of computation 2 and 3 were omitted and a narrow-edge detecting filter was employed at level 5. Thus, both contrast enhancement and feature detection can be varied, not only for the entire image, but for each matrix within the image, under control of either a remote human operator or a local command computer. This position-by-position control of the processing of the image, represented by the leftward



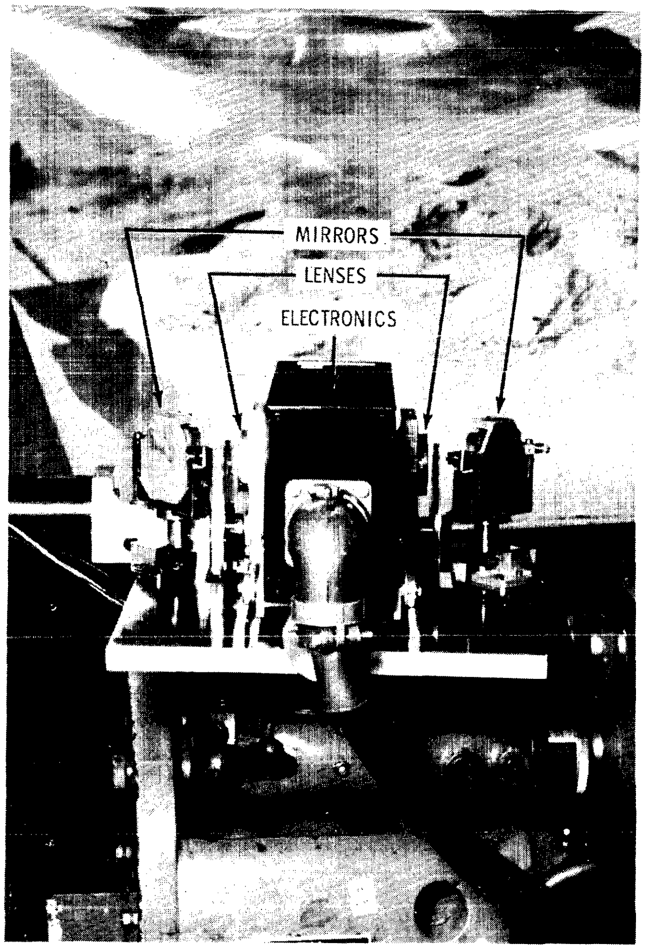


Figure 7—Equipment for simulating light-weight, low-power hardware; (above) Camera-computer chain for simulating visual computers, (below) binocular, or stereo, TV camera

arrows in Figure 1, separates our work from that at JPL. It makes perception possible.

Line drawings, such as those in Figures 5, 6 and 8, do not convey enough information for a scientist on earth to judge what is being pictured. However, by the addition of low-resolution luminance data to left and right views, and presentation of the two views stereoscopically, there may be enough information. Figure 10 is an example of levels 5 and 6 computation alone on data received from the scene pictured in

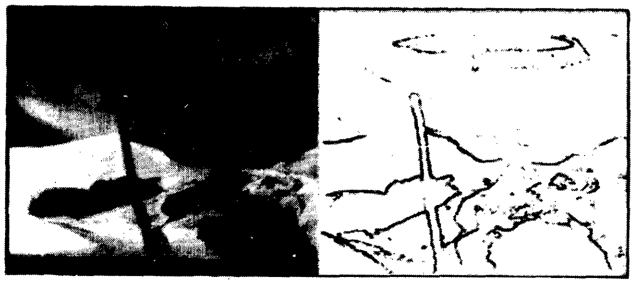


Figure 8—Formation of line drawing by levels 6 and 7 computation alone. Left-eye views of (left) image on TV monitor before being digitized, and (right) negative of scope display at level 8

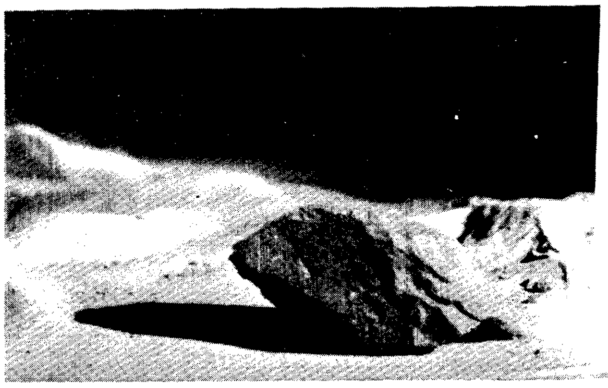


Figure 9—Photograph of scene to be formed into line drawing with shading

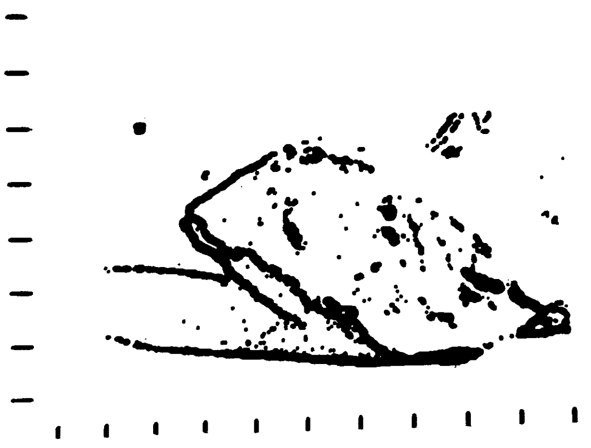


Figure 10—Computer-generated line representation of scene

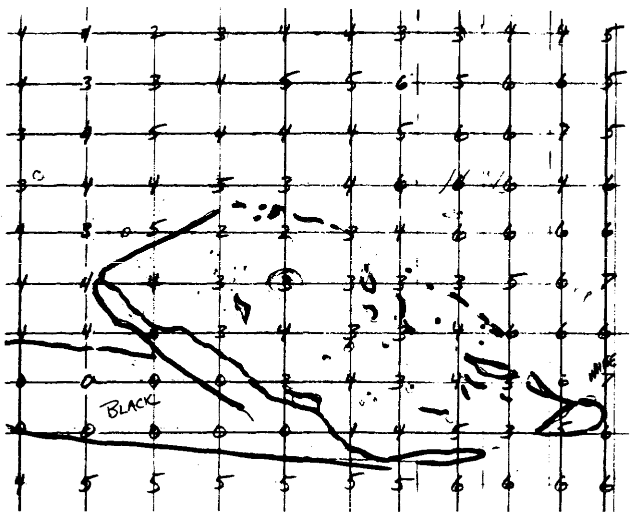


Figure 11—Grid of scene luminance values superimposed on Figure 10

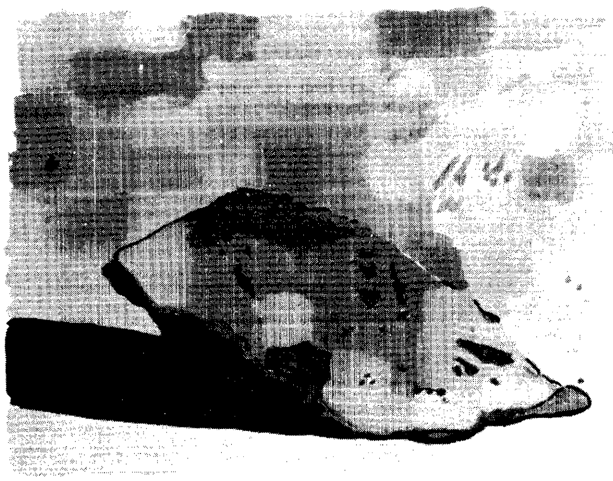


Figure 12—Painting with shades of gray prescribed by lines and values of Figure 11

Figure 9. Figure 11 shows coarse-resolution measurements of luminance on a scale from 0 to 7, which an artist employed to paint in the swatches of gray in Figure 12. When this reconstruction is performed by computer, it will illustrate how the appearance of a Martian scene can be reduced for transmission from Mars to earth and then reconstructed on earth for viewing there. The data reduction here is by a factor of 30.

The stereo pair of views, shown being reduced in Figures 5 and 6, was not taken with the mirrors illustrated in Figure 7 but by taking one at a time, moving the camera between takes. This is simpler for a report.

Hardware version of visual first-stage computer

We designed the computation first so that it could be implemented in light portable hardware; then we simulated this hardware in the computer of Figure 7 and achieved the results described above. The hardware design, diagrammed in Figure 13, is inspired by the layered structure of the animal retina and laterial geniculate body of the thalamus. Since it is not practical to represent in hardware the large number of cells of these animal structures, only a cluster of cells of each layer is represented and data are moved past the cluster by shift registers.

A camera containing a single vidicon, that receives left and right views from mirrors, is shown in preference to two cameras because the former arrangement leads to much less uncertainty between the two optical paths.[10]

In Figure 13, left and right images of an object such as a rock are projected by the optics onto the face of the vidicon. Converted to digital form, the signals enter shift registers (1) which move the images past computing elements (2) and (3), which represent clusters of living cells. The filter planned for level 2 is shown schematically at the upper left of Figure 13 and is given in detail in Appendix A. It consists of a central weight strongly positive, immediately peripheral negative weights and more peripheral weakly positive weights.

The images are advanced from left to right in the shift registers at the same rate that the tip of the electron beam in the vidicon advances. As data reach the right end of the top row, they are transferred to the left end of the next lower row. In this illustration, when the electron beam of the camera tube has swept 13 lines of the raster, the shift registers in the bank are full. From then on, for each new position of the electron beam, computations take place in the box behind the shift registers, and one digital word is discarded from the right end of the 13th row of shift registers.

Figure 14 shows test hardware under construction to perform levels 1, 2 and 3 computation, on five lines of the raster. Each of the lower five panels contains a shift register 6-bits deep. The registers shift the data past the computing element in the top panel. The medium-scale integrated circuits to be employed here, together with their wiring, can be packed into about 50 cu. in. With large-scale integrated circuits the volume could be 10 cu. in.[10]

Computation of range

To perform second-stage computation, either a man or a robot needs a view from a second position. We refer to the two views as "left" and "right" since they are usually taken from the same horizontal baseline. If the levels of robot computation pictured in Figure 4 are for the left view, then either a second series of levels is needed for the right view or the levels need to be widened to accommodate both views. We have taken the latter approach in the design of the hardware.

Animals compute range from comparisons of left and right images, at several levels of computation, and from the angle between the axes of the two eyes,

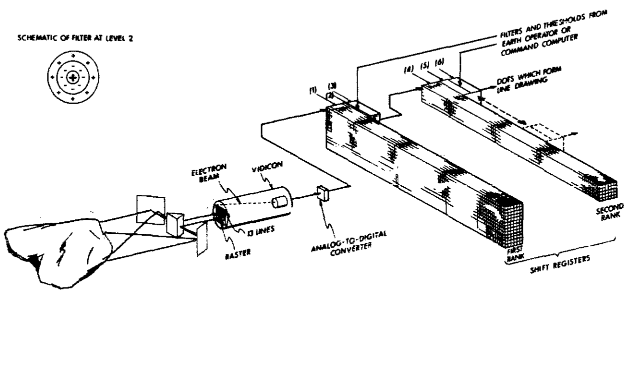

SCHEMATIC OF FILTER AT LEVEL 2
FILTERS AND THRESHOLDS FROM EARTH OPERATOR OR COMMAND COMPUTER
(1)
(2)
(3)
(4)
(5)
(6)
DOTS WHICH FORM LINE DRAWING
ELECTRON BEAM
VIDICON
13 LINES
RASTER
ANALOG-TO-DIGITAL CONVERTER
SECOND RANK
FIRST RANK
SHIFT REGISTERS

Figure 13—Diagram of hardware to perform contrast enhancement

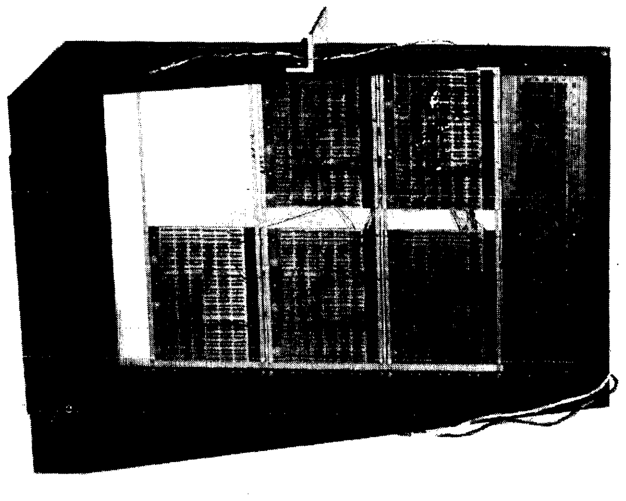


Figure 14—Visual first-stage computer under construction. The top panel is the computing element. The other five are shift registers. Integrated circuits plug into the far side

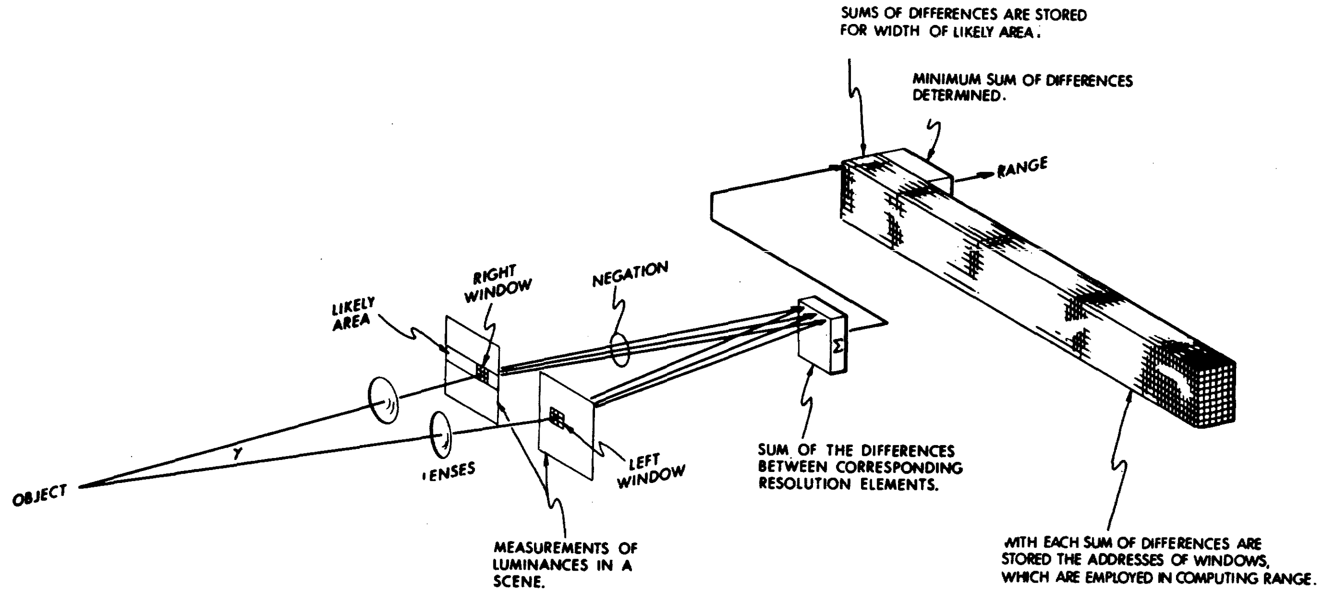

SUMS OF DIFFERENCES ARE STORED FOR WIDTH OF LIKELY AREA.
MINIMUM SUM OF DIFFERENCES DETERMINED.
RANGE
RIGHT WINDOW
NEGATION
LIKELY AREA
Σ
γ
OBJECT
LENSES
LEFT WINDOW
SUM OF THE DIFFERENCES BETWEEN CORRESPONDING RESOLUTION ELEMENTS.
MEASUREMENTS OF LUMINANCES IN A SCENE.
WITH EACH SUM OF DIFFERENCES ARE STORED THE ADDRESSES OF WINDOWS, WHICH ARE EMPLOYED IN COMPUTING RANGE.

Figure 15—Comparison of left and right views to determine range

called the "convergence angle."[11] The equipment shown in Figure 7 has been programmed to automatically compute range by comparing areas, in the left and right level 2 images, called "windows" (see Figure 15). To find a pair of windows that are viewing the same object, a window is first fixed on say, the left view, according to some criterion such as the presence of an edge; then a likely area is located in the other view. Since this area contains as many potential windows as resolution elements along the horizontal axis, the problem is to determine which of these windows corresponds to the one fixed in the left view. The simplest way is to compare luminances in the fixed window with corresponding luminances in the likely area, determine the difference and use this as a criterion to decide when a best match is obtained. From the data of the best match, range is computed by triangulation.

Employing this method, equipment shown in Figure 7 automatically explores a likely area to determine the range of an object at 20 feet with an uncertainty of 1.5 percent. To perform the comparison over the entire likely area requires 16 seconds. The comparison will be performed over less area if the robot visually follows around the edge of an object or visually explores increasingly deep into the scene. In these latter cases, the visual subsystem of the robot starts with a known range and reaches out from it.

Perception

In the robot we plan, the command computer, assisted by the relational computer, will determine what is seen by setting filters and thresholds in all stages of visual computation so as to match an iternally-generated image with an incoming one. This "Keeping up to date the internal organizing system, that represents the external world, by an internal matching response to the current impact of the world upon the receptive system" is called "perception."[12] "In a sense, the internal organizing system is continually making fast-time predictions on what is going on out there and what is likely to happen out there, and it takes anticipatory action to counter the small errors that might threaten its overall stability."

A line drawing with shading, transmitted to earth for a scientist to view, aids his perceptual process, giving him clues about the presence of objects about which he can then demand more information. Within a robot, however, a line drawing with shading will be the result of interaction between the relational computer, setting filters and thresholds, and the second stage visual computer where these filters and thresholds are tried on incoming data. When equiped to perceive, a robot will make fast-time predictions, possibly as detailed as the computer-generated image of Figure 16. Our general purpose computer formed Figure 16 from the equation of a cylinder, its diameter, height and illumination. It appears that perception of the cylinder could take place in the first and second stages of visual computation if the filters there are continually changed, as data are shifted past them, to search for predicted luminances in each part of an internally-generated image.[13]

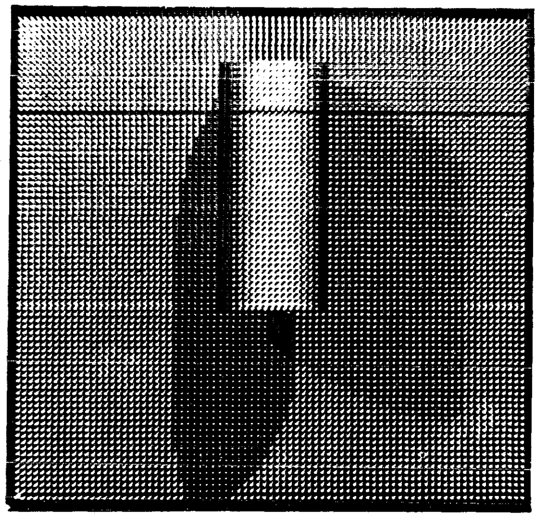


Figure 16—Picture generated by computer preparatory to experiments in perception

The concept of a command computer

The purpose of a command computer, in an animal or robot, is to commit the entire system to one of a dozen or so mutally exclusive modes of behavior. An animal requires such a computer because it cannot "fight," go to sleep, run away, and make love all at once."[14]

Our study of a possible Mars robot indicates that it, too, can only enjoy one mode of behavior at a time. Possible modes of such a robot are:

1. Look and feel
2. Advance
3. Retreat
4. Right itself if overturned
5. Maintain itself
6. Chart its environment
7. Rest

Perform incompatible experiments as follows:

8 Experiment A
9. Experiment B
10. Experiment C

"Look and feel" is a separate mode from "advance" because the robot must be stationary while either its camera or its arm is employed.

By "chart its environment" we mean that the robot, after advancing (or retreating) an appropriate distance, will establish what a surveyor calls a "station," and mark it with a transponder. Having determined the distance from the previous station and measured the angle between this and the second previous station, the robot can form a triangle in its memory to add to other triangles previously formed. By building triangle upon triangle, the robot establishes a grid with which it can determine how far it has gone and in what direction. Through this grid it can later pass, to return to points it has already visited.

Within each mode are the detailed sequences we call acts. Advance, for example, can be either slow, fast, or by turning. These details can be developed after the command computer has been designed to choose among the above modes.

The command computer should be capable not only of selecting the mode of behavior, in response to inputs from the environment, and the other computers, but of changing the way it responds to those inputs. Ability to change the record of conditions under which it selects a mode is called "plasticity" and is exemplified by conditioning and habituation.

The first simulation of a command computer represented only its mode-selecting ability. It was called S-RETIC where S stood for its ability to respond to both its present internal state and its spatially structured input.[15] The second simulation is called STC-RETIC where T stands for its ability to be influenced by temporal aspects of the environment and C for conditioning and habituation. The properties of STC-RETIC together with several new features are now being designed into special hardware to be called H-RETIC.

The inputs to each of these RETICs represent connections from such subsystems as the visual, described in part above, and the contact, described in part in a later section. The number of input channels to the present RETICs is very much smaller than will be required eventually from, say, the visual subsystem of the robot. In fact, the number of input channels, (γ_1 to γ_{42} in Figure 17) is only as many as needed to demonstrate the principles of operation.

How an animal is commanded

In the core of the reticular formation of animals (retic), the selection of a mode of behavior is made by a matrix of fan-shaped cells embedded in regions that are stacked like poker chips. The input processes of each cell, its dendrites, give it both its fan-shaped and its poker-chip-like thickness. Its output is an axon which traverses the long axis of the chip-like stack. Each cell in general receives signals from several parts of the brain, from many other reticular cells, especially those in its own neighborhood, and from several interoceptive and exteroceptive sensory-pathways. Collectively, these cells decide which mode of behavior the animal will enter. In its sharpest form,

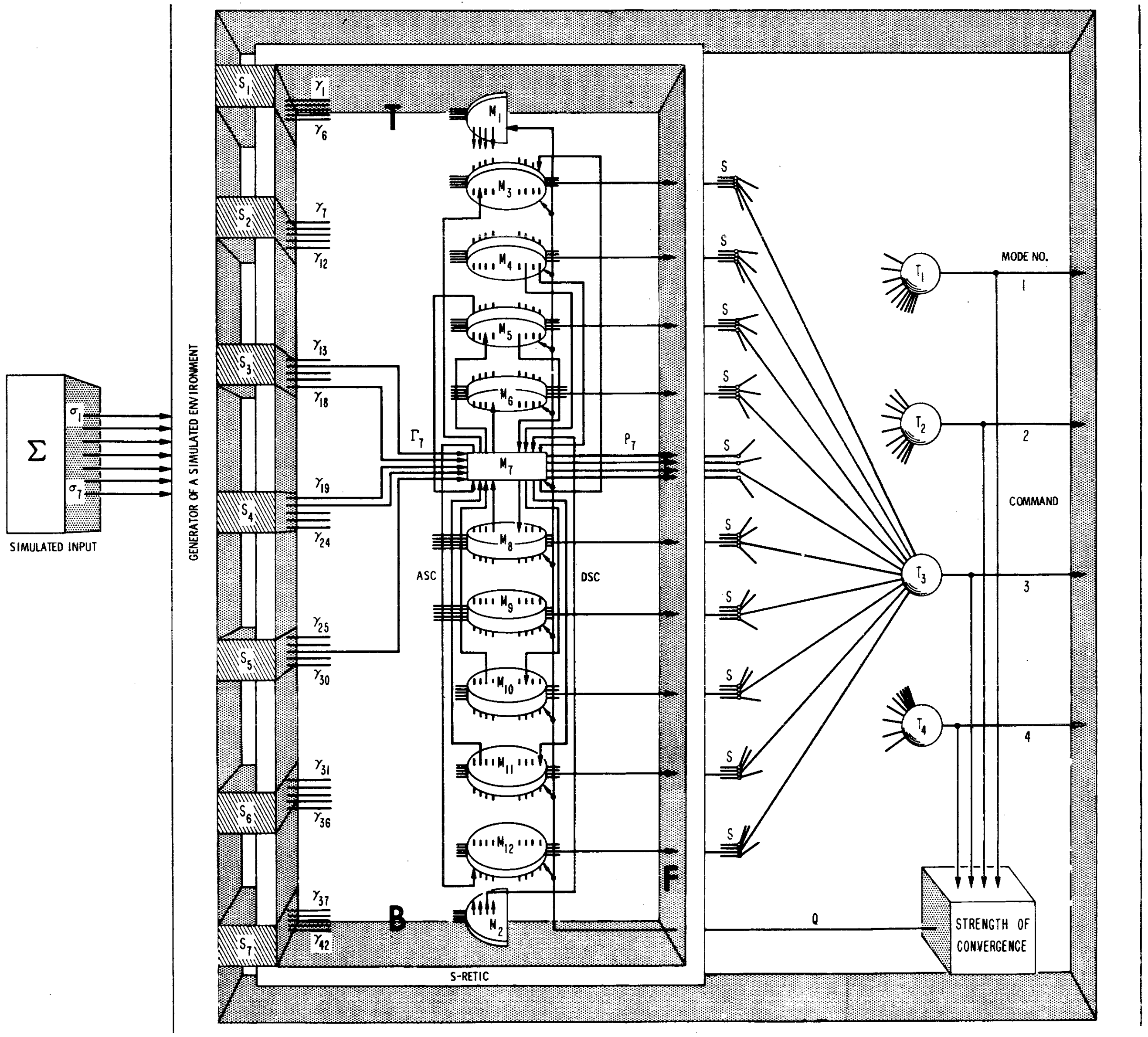


Figure 17—Simulation model of the command computer, S-RETIC, and threshold units (T_j) that determine convergence. The M_j are logic modules; the S_j are sensory systems; all P_j (only P_7 shown) are modular mode-indicating output lines; Σ simulates a RETIC environment that engenders input signals (sight, sounds, etc.); T and B are the top and bottom boundaries of S-RETIC; asc and dsc are the ascending and descending "nerve" bundles. For clarity, the connections that recur on all M modules are shown only on M_7

this assertion is only a hypothesis; but broadly speaking it is an evident biological fact.

The informational organization of retic is analogous to a board of medical doctors who must decide upon the treatment each of a series of patients must receive. Suppose there are twelve doctors on the board each a generalist, as well as a specialist in a different field of medicine, and that they must select one of four possible treatments. Their deliberations resemble the process by which S-RETIC selects a mode of behavior.

Like the board of medical doctors, the command computer (retic) must commit its charge to a mode of behavior which in most cases is a function of the information that has played upon it only over the last second or so (signals indicating mission are part of this). It receives information that is vast in amount, but highly correlated between input channels and arrives at one of a small number of mutually exclusive modes of behavior in a dozen or so time steps, with minimal equipment and maximal reliability. After a mode is decided upon, it must send out control directives to the other agencies in its charge to turn them to their respective tasks. Finally, that part of the command computer which at any given time has the most crucial information has authority over the mode of operation.

First simulation of a command computer, S-RETIC

Like retic, S-RETIC resembles a stack of poker chips, but each chip is now a computer module, M_i, and represents many retic cells (see Figure 17). Together the modules of S-RETIC decide on a mode of behavior in response to data from an overall environment that is static while each decision is being made. Note the word "overall". A major part of the environment of retic is the cerebral cortex where are stored the plans and goals of the animal. Although S-RETIC has only 42 binary input lines and chooses among only four modes of behavior, it demonstrates principles that can be applied on a much larger scale for a robot.

The 42 input lines, λ_i, are connected from sensory subsystems, S_i, to modules, M_i, in several-to-several, but not all-to-all fashion. The outer box in Figure 17 represents a generator to simulate an environment formed in response to the input, Σ. At this stage of design, all σ_i and γ_i lines carry binary signals. All of the other lines into and out of modules carry numerical estimates of probabilities.

Each of the 12 logic modules computes from its input information the probability that each mode of behavior is best for the robot. After this initial guess, the modules communicate their decisions to each other over the low capacity ascending and descending lines. Then each module combines this information in a nonlinear fashion with new information coming into it to arrive at adjusted probabilities that each of the four modes is best for the robot. The module, in turn, communicates this adjusted guess to the modules to which it is connected above and below. The process repeats until six or more modules find one mode of behavior best for the robot with a probability greater than 0.5. This threshold is sensed by threshold units T in remote motor controls.

Each try at consensus, or convergence, is called a cycle of operation. For details see Appendix B.

S-RETIC is a computer program operated now as part of the larger program described below. It always converges to the desired mode of behavior in less then 25 cycles, but 30 are allowed for it in a larger time period called an "epoch" with the new model described below.

Second simulation of a command computer, STC-RETIC

In the second simulation of a command computer, the already-operating S-RETIC was expanded to provide interconnections between the a-parts of the modules (ω lines), short- and long-term memories (STM and LTM) in each a-part and channels through which the experimenter can reinforce each module. In addition, the number of γ lines to each module was increased to seven; the σ lines were increased to 13, and the mode of behavior of the robot (RM) was fed back to each a-part.

The new model is called STC-RETIC where S and T stand for a spatially and temporally changing environment, C for conditioning and habituation. Where each module of S-RETIC has a transformation network to form its initial guess, each module of STC-RETIC draws its initial guess from its LTM. During this and the ensuing epoch, it computes the effects of reinforcements, given it by the operator, and then adjusts its LTM accordingly. The result is conditioning, habituation and other forms of "plastic behavior". For details see Appendix C. Given there are examples of the Pavlovian conditioning of a robot in a remote environment and the dropping out of this conditioning, called habituation. Development is also presented there as a form of plastic behavior.

The next step is to design a computer to be both a refinement of STC-RETIC and a more faithful reflection of the neurology of the core of the reticular formation. H-RETIC as it will be called, where H stands for hardware, will be organized much like the STC-RETIC with physically separate modules containing adaptive elements for memory.

Contact subsystem

Design of a sense of touch has progressed to the point of planning a hand and arm to reach out and press lightly against surfaces to the front and side of the robot. Figure 18 shows the tactile hand with a single finger. A grasping hand is also being designed to be carried by a second arm.

As shown by Figure 18, the shoulder provides three degrees of freedom: linear extension and rotations in elevation and azimuth. All motions are polar-centric, that is, centered on a common point, so that transformation of coordinates can be as simple as possible.

It is planned to map the probings of the finger in a somatic first-stage computer, similar to the visual first-stage computer described previously. The z-axis of that mapping will be depth rather than luminance. Sudden changes in depth will be detected by lateral inhibition, as they are in animals.[8]

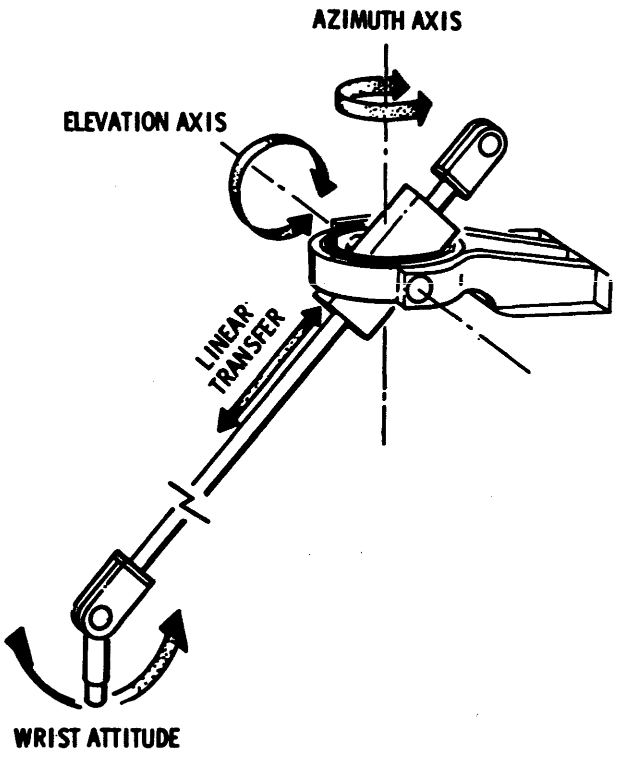


Figure 18—Degrees of freedom of the arm and hand

Command of the commander?

Since both visual and tactile computation can be commanded to respond to some sizes and shapes more strongly than others, we are led to ask: What commands the command computer of an animal? Dr. McCulloch's answer is revealing: "Nothing. You can persuade or cajole the command computer, but you cannot command it." His statement is supported by his diagram in Figure 2 where the influences upon retic are represented by the many connections to it. "There need to be connections from more than one sense, preferably at least three," he continues. Other influences upon retic are internal, such as the cerebral cortex, where are stored models of the environment, past and present, and future models in the form of goals. These influences may be stronger than external ones. Influences to a model of retic are represented by the S_i of Figure 17.

Modes, which the retic of a vertebrate animal chooses among, are as follows.[16]

1. sleep
2. eat
3. drink
4. fight
5. flee
6. hunt for prey or fodder
7. explore (or search)
8. urinate
9. defecate
10. groom
11. mate (or sex)
12. give birth or lay egg
13. mother the young (e.g., suckle, hatch, retrieve)
14. build or locate nest
15. innate forms of behavior unique to the species, such as
    - migrate
    - hibernate
    - gnaw
    - hoard

A comparable list for a Mars robot is given in a previous section.

We can imagine how a man's retic selects among his possible modes of behavior. Vision consists of both the feed-inward of raw sensory data and the feed-outward of signals to adjust filters to match internally-generated images. Touch, hearing and kinesthesia appear to operate in similar fashion. Sensory inputs, therefore, of the kind represented by the S_i of Figure 13, represent information from external and internal sensors and internal computers. Consider the actions of a soldier on sentry duty at an advanced post in enemy country. He is expected to hold his filters to match the stimuli he has been taught to expect from the enemy: the shape of his face, the color of his unifoım, his manner of fighting, etc.[17] If the soldier hears the snap of a breaking twig, turns toward the sound, and receives from its direction images that match the projections of his stored models, he may classify the sound and the sight as "enemy." What the soldier does next depends upon other models stored in his relational computer. If circumstances appear to favor combat, the sentry may fight (mode 4). If circumstances appear overwhelmingly adverse, the sentry may turn and run (mode 5). Circumstances that cause his command computer to select a mode are the number of the enemy, its armament, etc., as these are recognized by the process of generating projections and comparing these at the filters with incoming images. Thus the enemy can persuade or cajole the soldier's command co nputer.

Similarly, the Martian environment should be able to persuade or cajole the command computer of a Mars robot, in cooperation with very many fewer relations than are required for a human being, such as a sentry. As far as we know the Mars robot will not have to contend with enemies and, therefore,

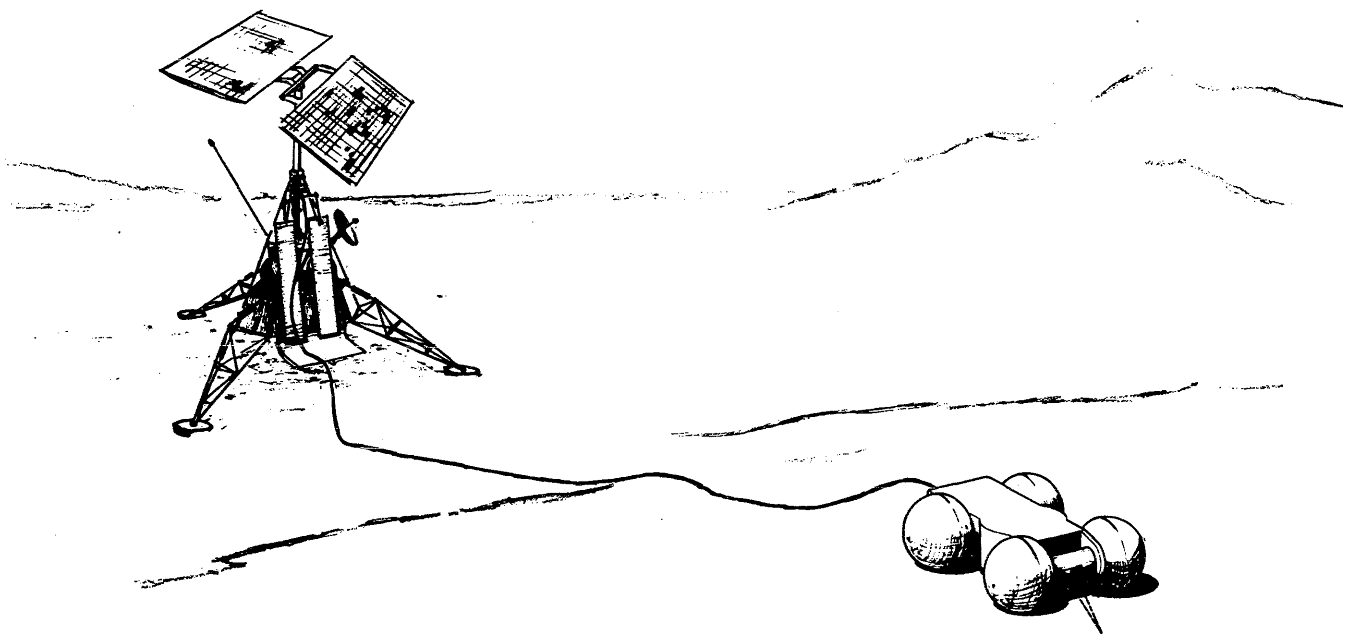


Figure 19—Proposed mobile data-acquiring element for a Mars landing

will not have to fight or flee. Nor will it have to enter any of the other modes which an animal has to enter in order to eat and reproduce.

The ten modes suggested in a previous section are very much reduced versions of the requirements of an animal. Instead of the elaborate mode of behavior to hunt, are the much simpler ones to advance, retreat, and look-and-feel.

Stages in the development of a robot

The system we have just described can be achieved, we believe, by such a process of designing, simulating, and testing as we have described for the development of the separate computers.

A camera-computer chain of the kind described previously can be packaged so as to be mobile. A possible mobile camera-computer chain is shown in Figure 19. To eliminate the need for its own power supply and transmitter to earth, it is connected to a Mars lander by a cable which it pays out from a spool at its center. To permit it to look from side to side without moving, its camera employs a rotating prismatic mirror which reflects the light of the scene through both left and right optical paths, to photocells which transduce luminances to voltages. The voltage amplitudes, when converted to digital words, then enter shift registers, to move past computing elements in the manner described earlier. Vertical scan is attained by turning the drum that holds the camera.

In initial experiments, the visual first-stage computer can be at the far end of the cable in the lander. When a light-weight visual first-stage computer has been completed, it can be placed within the mobile data-gathering element.

The command computer will not be tied in initially, the command being exercised by the earth operator. When the command computer has reached the stage of development where it can be tied in, it can be built into a robot that is physically separate from the lander, in a configuration such as that shown in Figure 20. This design shows, for test purposes, a stereo television camera mounted in gimbals on a commercially-available tractor. The gimbal mounting of the camera permits it to look forward, sideward, up, down and backward. Besides the camera is the arm described in an earlier section. The rubber tires would be replaced for Mars travel by wire-mesh tires.

Comparison with other robot projects

Perhaps because it is directed toward development of the properties of the computers described in an earlier section, ours is the only robot project with a binocular TV camera as input, a visual second-stage computer to employ this binocular input for precision computation of range, and a command computer to receive

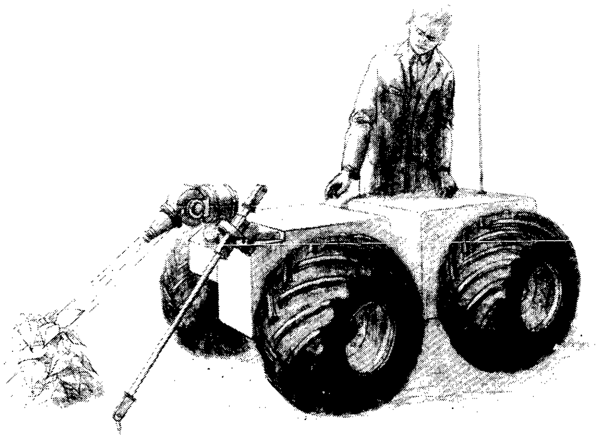


Figure 20—Proposed experimental robot

simultaneous inputs from several senses and decide what class of thing the robot should do.

However, the visual computers and the command computer are still operating separately while in other robot projects assemblies of computers and external equipment are already operating together. Eye-computer-arm-hand systems are in operation at Project MAC, M.I.T.[18] and the Department of Computer Science at Stanford University.[19] A computer-arm-hand system is in operation at the Department of Mechanical Engineering, M.I.T.[20] An eye-computer-vehicle system is in operation at Stanford Research Institute (SRI).[21] Out of the efforts of the projects have come list processing langugaes[22] [23] which we may use in simulating a relational computer. Other contributions are speech recognition,[19] the kinematics of manipulators under computer control,[24] the mapping of the space in which a manipulator operates[25] and recognition of visual contiguity.[26]

Since we are all engaged in processing increasingly large volumes of data, reward goes to the one who discovers a method of extracting useful data. At Project MAC and Stanford the reward can take the form of a doctor's degree; at SRI and our project, which are more equipment oriented, the reward is to have either the equipment or a simulation of it work. This reward is also present at the two universities; and SRI and we also do theoretical work.

Aside from the amount of equipment in operation, the greatest difference between our project and the other three is in the way we use the life sciences. All of us use this information since we are all trying to make computers and other hardware do what until now, only animals have done. However, we give primary attention to anatomy and physiology, secondary attention to psychology, while other projects assign reverse priority. We make an exhaustive search of the literature on each animal "computer" we investigate and attempt to create, within the constraints of technology, a working model of the "computer". Examples of such literature searches are given in References 28, 4 and 16.

Once we have achieved a working model we proceed to improve it and in doing so may excel nature. We are now reducing the appearance of a scene to stereograms that are much better than Figure 6, which had resulted from out attempt to model animal vision.

CONCLUSION

To some, the work reported here will appear to be slavish imitation of the vertebrate nervous system. It appears to us, on the other hand, to be good engineering practice. When something does what you want it to do and is already miniaturized; why not copy it?

The copy is made only in principle. Visual computers are time-shared elements, past which data is moved by shift registers. The modules form larger portions of the command computer than do cells in retic. In fact, the modules and the cells are similar only in their mathematics.

To quote Dr. McCulloch, "We use the word 'robot' in two ways. The first, and less important, is as a machine that performs isolated functions of a human being. The second, and more important, is as a description of life applicable to either living things or machines. Such a description is indifferent to whether the system is man-made or grown, whether it is built of hardware or living cells. This is a central theme of cybernetics: to use the same logic and mathematics to describe men and machines. Norbert Wiener looked at control this way. We are looking at both command and control. Thus, in the more important sense, a robot is a prescription for a system that until recently could be achieved only by the growth of living cells but is becoming something we can manufacture."

ACKNOWLEDGMENT

Because the physiological concepts presented here came from Warren McCulloch it would have been correct to list him as an author, but in doing so we could not have identified his contributions. One who helped us describe these contributions was Dr. Jerome Y. Lettvin. The simulation of the visual first-stage computer is the work of James Bever who began it, John Hatfield who devised the form of

filter employed in level 5, and Jerome Lerman who devised all of the filters presently employed. The two simulations of retic were carried out by Jay Blum.

REFERENCES

1 C M MICHAUX
Handbook of the physical properties of the planet Mars
NASA 1967 for sale by the Superintendent of Documents
U S Government Printing Office Washington D C 20402

2 SPACE SCIENCE BOARD OF THE NATIONAL ACADEMY OF SCIENCES
Planetary exploration 1968–1975
2101 Constitution Avenue N W Washington D C 20418

3 L L SUTRO
Information processing and data compression for exobiology missions
R–545 Instrumentation Laboratory M I T
Cambridge Massachusetts 1966

4 R MORENO-DIAZ
An analytical model of the group 2 ganglion cell in the frog's retina
Report E–1858 Instrumentation Laboratory M I T
Cambridge Massacuhsetts 1965

5 W S Mc CULLOCH
Embodiments of mind
M I T Press Cambridge Massachusetts 1965

6 D DENNY-BROWN
The cerebral control of movement
Chapter VI Charles Thomas Publishers 1965

7 R H SELZER
The use of computers to improve biomedical image quality
Proc F J C C 1968

8 G VON BEKESY
Sensory inhibition
Princeton University Press Princeton New Jersey 1967

9 F RATLIFF
Mach bands: quantitative studies of neural networks in the retina
Holden-Day Inc San Francisco 1965

10 L L SUTRO C D SIGWART J D LEAVITT D G TWEED
Instrumentation of camera-computer chains
Report R–636 Instrumentation Laboratory M I T
Cambridge Massachusetts 1969

11 R GREGORY
Eye and brain
World University Library McGraw-Hill New York 1966 50–60

12 W M BRODEY N LINDGREN
Human enhancement beyond the machine age
IEEE Spectrum Describing the work of Donald McKay
February 1968 89

13 L L SUTRO
Computer synthesis of images with shades and shadows
Report E–2250 Instrumentation Laboratory M I T
Cambridge Massachusetts 1969

14 W L KILMER W S Mc CULLOCH
The reticular formation command and control system
Proc Symposium on Information Processing in the Nervous System, State University of New York at Buffalo October 1968 (in pubication)

15 W L KILMER W S Mc CULLOCH J BLUM L SUTRO
A model of life to select a mode of behavior
Report R–635 Instrumentation Laboratory M I T
Cambridge Massachusetts 1969 (in preparation)

16 W L KILMER W S Mc CULLOCH
The biology of the reticular formation
Michigan State University Division of Engineering
Research Report East Lansing Michigan (in preparation)

17 J D FRANK
The face of the enemy
Psychology Today November 1968 25

18 M L MINSKY S A PAPERT
Research on intelligent automata status report II
Project MAC M I T Cambridge Massachusetts 1967

19 J Mc CARTHY L D EARNEST D R REDDY P J VICENS
A computer with hands eyes and ears
Proc F J C C 1968

20 W R FERRELL T B SHERIDAN
Supervisory control of remote manipulation
IEEE Spectrum Vol 4 No 10 October 1967 81–88

21 N J NILSSON C A ROSEN B RAPHAEL G FORSEN L CHAITIN S WAHLSTROM
Application of intelligent automata to reconnaissance
Stanford Research Institute Menlo Park California 1968

22 E C BERKELEY D G BOBROW editors
The programming language LISP
Information International 200 Sixth Street Cambridge Massachusettts 1967

23 L G ROBERTS
Machine perception of three-dimensional solids
Optical and electro-optical information processing
M I T Press 1965

24 D L PIEPER
The kinematics of manipulators under computer control
Computer Science Department Stanford University
Palo Alto California 1968

25 D E WHITNEY
State space models of remote manipulation tasks
Engineering Projects Laboratory Department of Mechanical Engineering M I T Cambridge Massachusetts 1968

26 A GUZMAN
Some aspects of pattern recognition by computer
MAC–TR–37 Project MAC M I T Cambridge Massachusetts 1967

27 ILLUMINATING ENGINEERING SOCIETY
IES lighting handbook fourth edition
345 E 47th Street New York 1966 2–6

28 L SUTRO editor
Advanced Sensor and Control System Studies 1964 to September 1965
Report R–519 Instrumentation Laboratory M I T
Cambridge Massachusetts 1966

APPENDIX A

Filters

Scaling

As indicated earlier, application of a filter such as

G_2 will produce matrices of luminances that are "on scale", except when applied to spots and edges of maximum contrast. By "on scale" we mean on the scale of 0 to 63 that our digital-to-analog converter can convert and our scope can display.

When filter $G_1 \times 1/8$ is convolved with all of the submatrices of luminance in a typical scene and a histogram plotted of the convolution sums, the result is Figure 21. The extremes are formed when the center of the submatrix of luminances is 63 and the surround is zero, or vice versa. By lopping off extremely high and low values of the convolution sum, as shown by cross hatching, the scale is contracted. Added to the central luminance of the submatrix in question, the convolution sum enhances contrast. In the rare case where this addition forms a negative sum, this sum is replaced by zero. Filter $G_1 \times 1/8$ is used in this example, rather than filter G_2, to demonstrate a symmetrical histogram. The 2 in the center of filter G_2 leads to the unsymmetry.

Filters to enhance contrast

Luminance contrast is defined[27] as

$$\frac{|L_1 - L_2|}{L_1} \qquad \text{(A1)}$$

Where L_1 and L_2 are the luminances of the background and object, respectively. Such contrast is determined in animals and machines, not by a simple subtraction and division as here, but by the convolution of a filter with the luminances in the image.

In previous sections we considered the application of a minimum-area filter to an image. Here we consider filters of any size that may be useful. If each filter, in a matrix of filters, has a sensitive central excitatory region surrounded by a less sensitive inhibitory region, possibly surrounded in turn by a still less-sensitive excitatory region, the matrix may be represented by the schema of Figure 22.[9] The pattern of response is here determined by the net effects produced by overlapping, and possibly opposed, excitatory and inhibitory influences. This interaction of influences may be expressed by a set of simultaneous equations, one for each filter.

Implied in the diagram of Figure 22 is mutual inhibition which can be represented by Figure 23 if the recurrent lines are headed by circles, instead of arrows, to represent inhibition. The effect of recurrent inhibition can be achieved either by such connections, by the non-recurrent filter shown in section in Figure 24, or by two applications of the 7 × 7 filter shown in Figure 24. Jerome Lerman devised this 7 × 7 filter which, when convolved with itself,

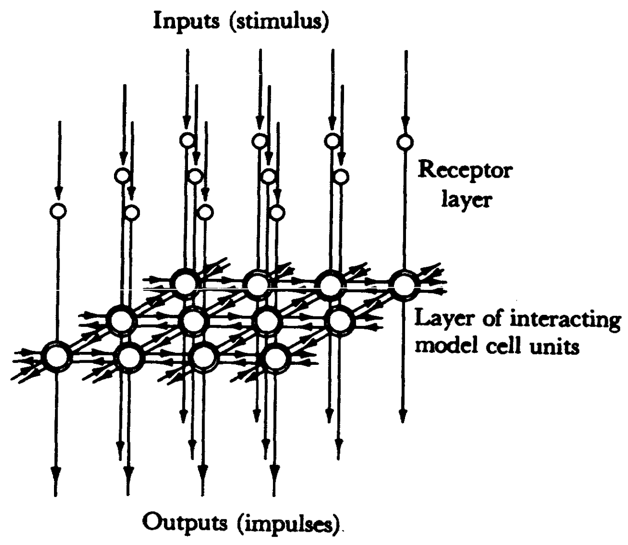


Figure 22—Schema of the receptive layer and interacting layer, with arrowheads indicating the direction of conduction of impulses in an animal, amplitude of signals in a robot

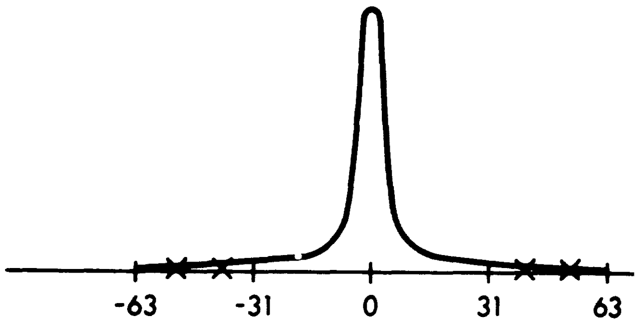


Figure 21—Histogram of convolution sums formed by filter G_1 with all of the sub-matrices of luminance in a typical scene

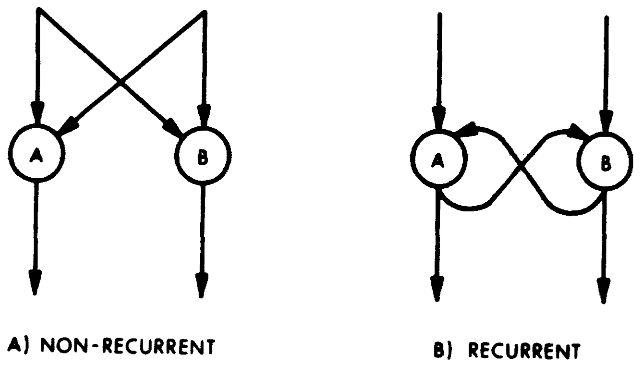
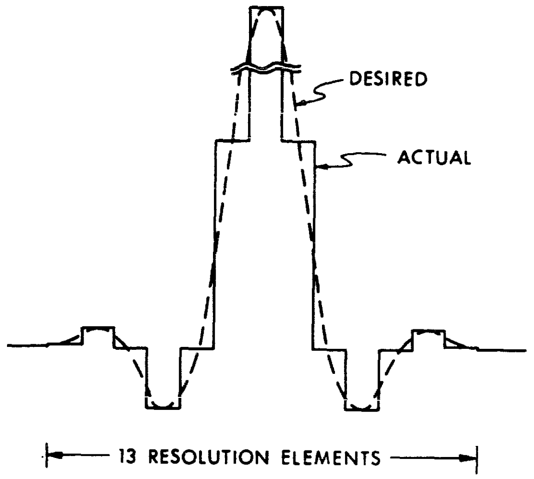


Figure 23—Interaction of cells A and B

a



b

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -2 & -2 & -2 & -2 & -2 & -1 \\ -1 & -2 & 6 & 6 & 6 & -2 & -1 \\ -1 & -2 & 6 & 64 & 6 & -2 & -1 \\ -1 & -2 & 6 & 6 & 6 & -2 & -1 \\ -1 & -2 & -2 & -2 & -2 & -2 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \bigstar \begin{bmatrix} \\ \\ \\ & & & \text{SAME} & & & \\ \\ \\ \\ \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 2 & 6 & 10 & 14 & 18 & 22 & 24 & 22 & 18 & 14 & 10 & 6 & 2 \\ 3 & 10 & 2 & -6 & -14 & -6 & -2 & -6 & -14 & -6 & 2 & 10 & 3 \\ 4 & 14 & -6 & -158 & -194 & -198 & -192 & -198 & -194 & -158 & -6 & 14 & 4 \\ 5 & 18 & -14 & -194 & -310 & -262 & -154 & -262 & -310 & -194 & -14 & 18 & 5 \\ 6 & 22 & -6 & -198 & -262 & 766 & 904 & 766 & -262 & -198 & -6 & 22 & 6 \\ 7 & 24 & -2 & -192 & -154 & 904 & 4472 & 904 & -154 & -192 & -2 & 24 & 7 \\ 6 & 22 & -6 & -198 & -262 & 766 & 904 & 766 & -262 & -198 & -6 & 22 & 6 \\ 5 & 18 & -14 & -194 & -310 & -262 & -154 & -262 & -310 & -194 & -14 & 18 & 5 \\ 4 & 14 & -6 & -158 & -194 & -198 & -192 & -198 & -194 & -158 & -6 & 14 & 4 \\ 3 & 10 & 2 & -6 & -14 & -6 & -2 & -6 & -14 & -6 & 2 & 10 & 3 \\ 2 & 6 & 10 & 14 & 18 & 22 & 24 & 22 & 18 & 14 & 10 & 6 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

Figure 24—(a) Dashed line, graph of cross-section of desired filter; solid line, graph of cross-section of approximation shown in (b). (b) Weights of the filter employed in enhancing the contrast of Figure 5

produces the 13 $\times$ 13 filter below it. If each 7 $\times$ 7 filter summed to zero, the center would be 8. By summing to 56 the filter enhances existing contrast. The 13 $\times$ 13 filter also sums to 56. The computer of Figure 7 applies the smaller filter twice since its core memory will hold only seven lines at a time of a digitized image, 256 positions wide with 6 bits per position.

The purpose of the diminishing waves of alternately excitatory and inhibitory effects in Figure 24 is to continue the spatial filtering outward from the center of influence. While this filter passes higher spatial frequencies and rejects low, the frequencies to which the vidicon responds are not very high, so it might be called a "middle-pass filter."

Filters to form a line drawing

To select points of high contrast that outline objects and details, the second-stage visual computer needs to take differences, not only to determine contrast as in formula A1, but to determine if this contrast exceeds a threshold.

Consider how these differences can be taken in scanning across the five bands of luminance, in Figure 25. The contrast across the area is represented by $\frac{|A - D|}{Av}$ where Av is the average of the five luminances. The threshold, for reasons explained below, is best expressed as the reciprocal of the inner difference B-C multiplied by a constant. Employing these differences, Jerome Lerman devised the following criterion which the computer applies first in the horizontal direction, then in the vertical.

If

$$\frac{|A - D|}{Av} - \frac{K}{|B - C|} > 0, \qquad \text{(A2)}$$

a dot is placed in the line drawing.

If

$$\frac{|A - D|}{Av} - \frac{K}{|B - C|} \leq 0, \qquad \text{(A3)}$$

no dot.

Figure 26 develops the reason for using the reciprocal of the inner difference, times a constant, as the threshold. Curve a is a ramp of luminance. Curve b is its derivative when Δx is the narrow span represented by the distance between 1 and -1 at the upper right of curve b. Curve c is the derivative of curve b when Δx is larger. To reduce curve a to a dot, it is evident that the threshold needs to be lowest when the derivative is both as great as possible and as peaked as possible. By plotting the reciprocal as in curve d, the peak does not go to infinity, as it could, and remains manageable. The threshold is a fixed fraction k of this reciprocal.

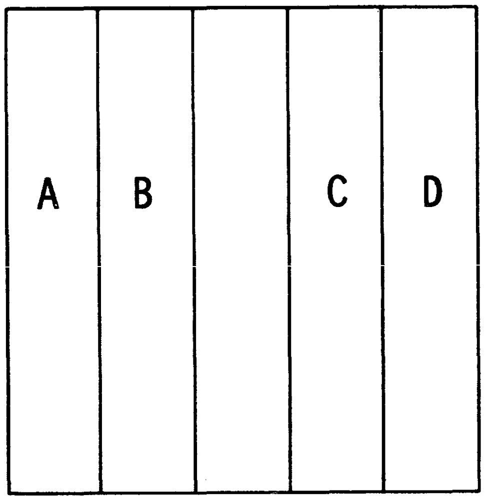


Figure 25—Four bands, each of approximately uniform luminance, about a band of luminance not lettered

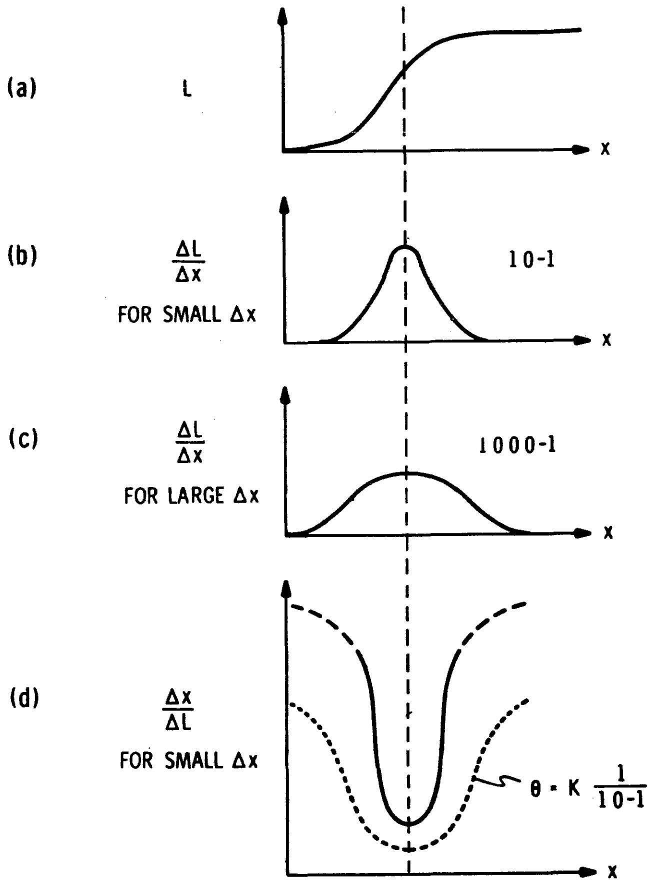


Figure 26—Explanation of use of reciprocal of inner difference, times a constant, as the threshold in determining whether a dot shall be placed in line drawing. See text for description of steps

APPENDIX B

Operation of S-RETIC

Each set of lines $\mathbf{p}_i$ in Figure 17 (only $\mathbf{p}_7$ is marked) indicates the ith module's degree of preference for each of the four possible modes. Thus $\mathbf{p}_i$ is a probability vector. The ascending and descending lines out of M_i are branches of $\mathbf{p}_i$ which connect to other modules.

As shown in Figure 27, each module has two parts. Each a-part computes from the module's input information the initial guess as to what mode of behavior is best for the robot. The b-part computes an adjusted guess from information received from above, below and from the a-part. The a-part which has five binary input variables and four probabilistic variables, is a nonlinear transformation network.

The b-part receives 4-component probability vectors $\mathbf{p}_\delta$ from above, $\mathbf{p}_\alpha$ from below and $\mathbf{p}'$ from the a-part. The components of each vector are the relative probabilities of each of the four possible modes of behavior. The jth component of each $\mathbf{p}_\alpha$, $\mathbf{p}_\delta$ or $\mathbf{p}$ probability vector is the probability, computed by the module of origin, that the command computer's present γ input signal configuration calls for mode j. The b-part also receives a measure of the cycle-to-cycle difference between γ inputs, called "γ difference" in Figure 27, and a measure, Q, of the strength of convergence on the last mode commanded (shown being formed in the lower right corner of Figure 17).

Consensus among the modules is achieved, as illustrated in Figure 17, by first determining in the step function (s), if the jth component of the probability vector $\mathbf{p}$ from each module exceeds 0.5. If it does, a 1 is passed on to the threshold element T_j. There, if 50 percent or more of the jth component input connections are 1's, mode j is commanded.

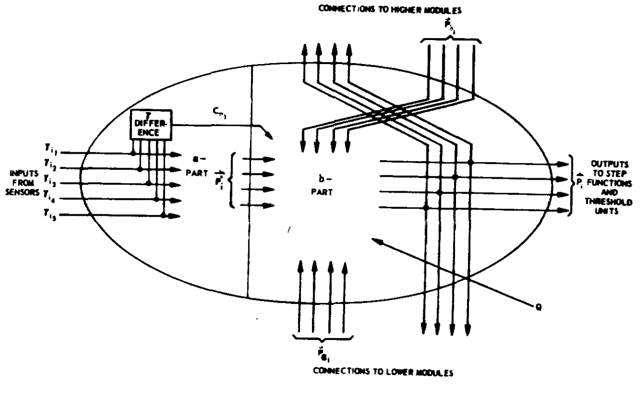


Figure 27—Input and output connections to parts a and b of a module in S-RETIC

Note that each element T_j in Figure 17 receives 10 inputs, although for clarity in the drawing, connections are shown only to T_3. The threshold elements T are part of the executive computer pictured in Figure 2. Each T receives many inputs, decodes them and executes a mode of behavior.

S-RETIC is described in more detail in Reference 15.

APPENDIX C

Operation of STC-RETIC

General

Figure 28 diagrams the a-part of a module in STC-RETIC. Signals from the senses (ω's), from modules above and below (γ's), from the threshold units that determine the mode (RM) and from the experimenter (RPC and RAC's) enter the short term memory where they are held for two input epochs. STC-RETIC converges on a mode in the same sequence described above for S-RETIC, except that each initial guess is read from a long term memory instead of being generated by a transformation network. Each initial guess is read out as a probability

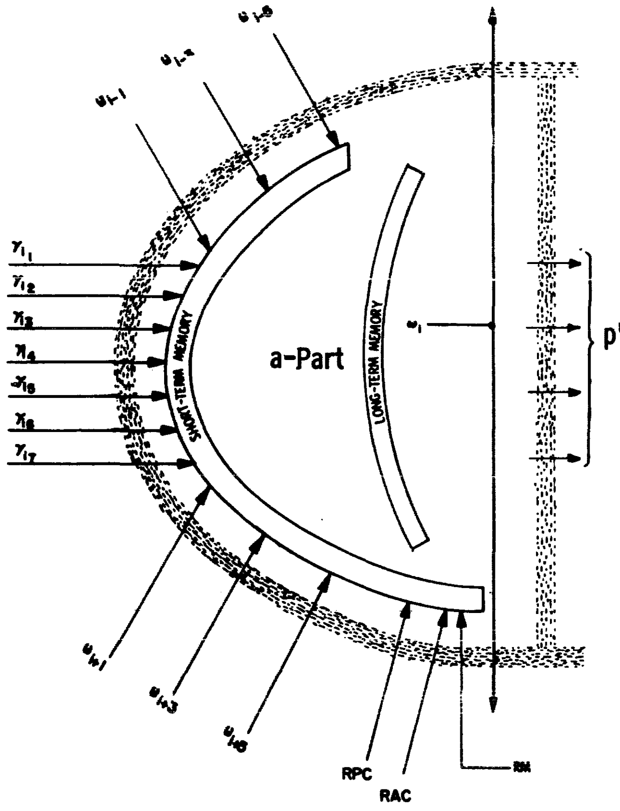


Figure 28—The a-part of a module in STC-RETIC

vector to be used in the same way as p'_i in S-RETIC (Figure 27).

In the conditioning process, the information as to module inputs, module p', the converged-upon mode and whether it was good or bad, is used to modify the probabilities stored in long term memory.

In either conditioning STC-RETIC, allowing it to habituate or in other ways encouraging it to acquire temporally influenced patterns of behavior, the operator of STC-RETIC not only provides the external conditions (Σ, Fig. 17) to which it will respond, but he takes the place of the parts of a robot that would indicate that the input and response conditions are punishing, neutral, or rewarding. He introduces a negative reinforcement prior to convergence (RPC) to indicate that the external conditions are themselves punishing, and a zero RPC to indicate that they are neutral. Finally he introduces four other numbers (RAC's) to indicate the reinforcement effect on STC-RETIC of its selection of each of its modes of behavior. An RAC_i is a "reinforcement to be applied after convergence if STC-RETIC converges to mode i." It is used in conjunction with short-term-memory information in the module to specify the corresponding modification to the module's long term memory. A negative value of RAC_i, called "punishing," is interpreted by STC-RETIC as evidence for not converging on mode i the next time the same γ's and ω's are received. A positive value, called "rewarding," is interpreted oppositely.

Examples of Pavlovian conditioning and habituation in the robot

Pavlovian conditioning is made possible both by responses "wired into" the long-term memory of each module of STC-RETIC, and by γ's and ω's acquired during two epochs by the module's short-term memory. For example, let us suppose that a "wired-in" (unconditioned) response is (1) to command retreat when the robot feels a horizontal edge with nothing beyond it (a precipice ?), then (2) to be rewarded (positive RAC) for saving the robot. Let us suppose further that each time the robot feels an edge, beyond which it feels nothing, it also sees an edge, beyond which it sees nothing. The more often the robot both feels and sees an edge, beyond which it feels nothing, and then is rewarded for retreating, the more firmly it becomes conditioned to back up at the sight, instead of just the feel, of such an edge.

In STC-RETIC such conditioning is stored in the long-term memory of each module, which contains a reduced record of the conditions under which it has made decisions.

An example of habituation in a robot is the following. Suppose a robot travels over the surface of Mars and comes upon territory that is sharply striped in light and dark. When it ceases to behave as though each stripe were a precipice, it will have become habituated to the striped terrain. After a time, the duration of which was pre-specified, it will spontaneously recover the original response so that, if it then moves into territory where such stripes are shadows marking true hazards, it will not be harmed. What have changed in both of these examples are the long-term memories of modules.

Forms of robot behavior due to development

As explained above, the initial guess of the a-part of each module is made by consulting its long term memory. If there is no entry for the given values of γ and ω, then a flat probability vector (all four components equal) is read out.

Changes in a long term memory, which starts from flat probability vectors, are what we mean by development. STC-RETIC has room in the long term memory of each module for 100 vectors, some of which are flat, others are preprogrammed to non-flat values. Some of the kind of development that is complete in the retic of a mammal at birth may be best achieved by STC-RETIC through experience in its environment.

Other forms of plastic behavior modeled by STC-RETIC are: generalization of and discrimination among the conditions under which a mode response is given, avoidance conditioning and extinction of Pavlovian conditioning.

Diagnosis and utilization of faulty universal tree circuits

by GIACOMO CIOFFI

Fondazione U. Bordoni
Rome, Italy

and

EUGENIO FIORILLO

I.B.M. Italia
Milano, Italy

INTRODUCTION

In the last few years the progress of integration techniques of more and more complex digital circuits has led to the development of a new branch of the switching theory known as cellular logic.[1] Nevertheless the integration of cellular circuits of a certain degree of complexity is hampered at present by poor yield, that is by the likelihood that one or more cells of the circuit may turn out to be faulty. It is useful, therefore, to try to use these circuits even when there are some faulty cells. Generally speaking there are two possibilities:

(a) to provide a certain degree of redundancy in the circuit, so as to be able to replace the faulty cells with spare cells;[2]
(b) to use, if possible, only the part of the circuit functioning correctly.

The second way seems easier to carry out, since it does not require interventions on the circuit already realized.

Apart from the choice of either way, it is necessary above all to carry out the diagnosis of the circuit in order to determine what failures have taken place.

In the present paper these problems are studied in the case of universal cellular tree circuits.[3,4] In the first part a minimal set of diagnostic tests is derived; in the second part a criterion is set forth which, on the basis of the results of the tests carried out, makes it possible to find some functions that can be realized by the faulty circuit.

Cellular tree circuits

A cellular tree circuit with n levels, A_n, is a circuit having the structure shown in Figure 1, and can implement any function of the n variables $x_1, \ldots, x_n$. The circuit is made up of 2^n-1 cells, all equal (Figure 2), which implement the function

$$c_j^i = c_{2j}^{i-1}\,\bar{x}_i + c_{2j+1}^{i-1}\,x_i\,.$$

The 2^n inputs $(\tilde{x}_1, \ldots, \tilde{x}_n)$* will be indicated briefly with $X = \sum_{i=1}^{n} \tilde{x}_i 2^{i-1}$. The binary inputs $c_0, \ldots, c_{2^n-1}$ are for the specialization of the circuit; to be exact each of the $2^{(2^n)}$ vectors $C \equiv (c_0, \ldots, c_{2^n-1})$ corresponds to one and only one function $F(x_1, \ldots, x_n)$. It is easy to show[3] that, if vector X is applied to A_n, the output c_0^n coincides with the specialization bit c_X; therefore there exists a one-to-one correspondence between the minterm m_j and the bit c_j $(0 \leq j \leq 2^n-1)$. A function $F(x_1, \ldots, x_n)$ is implemented on A_n giving the specialization bits corresponding to the minterms implicating the function values equal to 1 and leaving the rest at 0.

Tests to detect faults in an A_n

In view of the regularity and the simplicity of the

(*) $\tilde{x}_i = 0$ or 1

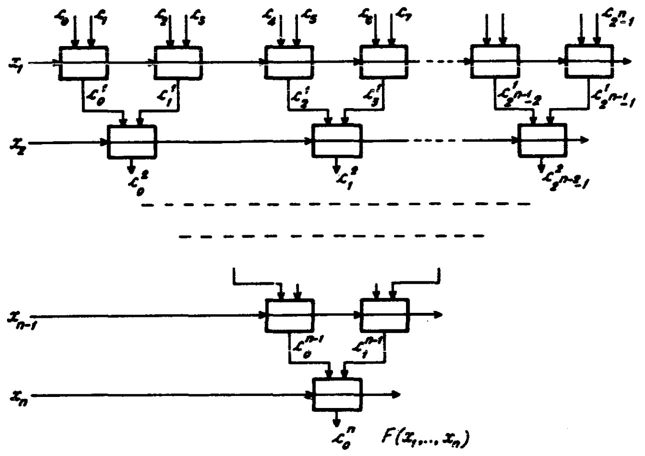


Figure 1—Structure of the universal cellular circuit (t.c.)

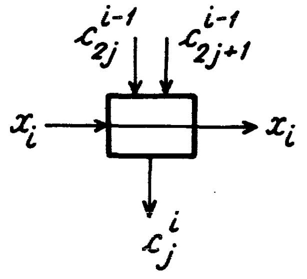


Figure 2—j-th cell on level i of the t.c.

interconnections, only faults within the cells are considered possible; this implies that, even when there are faults, the output of each cell always remains a function (possibly degenerate) of its three inputs exclusively. To remain on a more general plane, it will be supposed that a faulty cell can implement any of the remaining 255 functions of three variables.

This section describes a complete and minimal method of detecting faults in a tree circuit; that is, it detects all faults satisfying the above mentioned hypotheses and consists in the minimal number of tests.

Description of the tests

The test in which all terminals $c_0, \dots, c_{2n-1}$ are fixed at 0 and the inputs take on all possible values $X_0, \dots, X_{2n-1}$ is called here "fixed 0 test" (0_f). If the tree circuit (t.c.) functions correctly, during the fixed 0 test there is a sequence of 2^n zeroes in the output.

The test described in Table 1 is called here "travelling 0 test" (0_t). If the t.c. functions correctly, during this test there is a sequence of 2^n zeroes in the output.

The tests here called "fixed 1" and "travelling 1" (1_f and 1_t) are obtained from the 0_f and 0_t tests by complementing all the specialization bits. They give two sequences each of 2^n ones at the output of a correct t.c.

Let us consider the set of tests 0_f, 1_f, 0_t, 1_t applied to a correctly functioning t.c. The signals at every point of the circuit are shown in Table 2.

It can be seen that the four tests for the whole tree correspond to the same tests carried out on the subtrees and on the individual cells according to Table 3.

Let us take as an example a tree made up of only two levels (Figure 3).

The tests are those shown in Table 4, in which it is supposed that the circuit functions correctly.

It is then possible to construct Karnaugh's map of the function implemented by each cell (Figure 4); for the sake of clarity the maps have been arranged in the same way as the respective cells.

This result can be easily extended to the case of an n-level tree. On each level the squares of the maps are

Table 1

X	c_0	c_1	c_2	c_3	...	c_{2^n-2}	c_{2^n-1}
0	0	1	1	1	...	1	1
1	1	0	1	1	...	1	1
2	1	1	0	1	...	1	1
3	1	1	1	0	...	1	1
.....	.....	.....	.....	.....	.....	.....	.....
2^n-2	1	1	1	1	...	0	1
2^n-1	1	1	1	1	...	1	0

Table 2

X	$c_0c_1c_2c_3\dots c_{2^n-2}\ c_{2^n-1}$	$c_0^1c_1^1\dots c_{2^{n-1}-1}^1$	$c_0^2c_1^2\dots c_{2^{n-2}-1}^2$	...	$c_0^{n-1}\ c_1^{n-1}$	$c_0^n=F$
0	0 0 0 0 ... 0 0	0 0 ... 0	0 0 ... 0	...	0 0	0
1	0 0 0 0 ... 0 0	0 0 ... 0	0 0 ... 0	...	0 0	0
.....	.....	.....	.....	.....	.....	.....
2^n-1	0 0 0 0 ... 0 0	0 0 ... 0	0 0 ... 0	...	0 0	0
0	1 1 1 1 ... 1 1	1 1 ... 1	1 1 ... 1	...	1 1	1
1	1 1 1 1 ... 1 1	1 1 ... 1	1 1 ... 1	...	1 1	1
.....	.....	.....	.....	.....	.....	.....
2^n-1	1 1 1 1 ... 1 1	1 1 ... 1	1 1 ... 1	...	1 1	1
0	0 1 1 1 ... 1 1	0 1 ... 1	0 1 ... 1	...	0 1	0
1	1 0 1 1 ... 1 1	0 1 ... 1	0 1 ... 1	...	0 1	0
2	1 1 0 1 ... 1 1	1 0 ... 1	0 1 ... 1	...	0 1	0
3	1 1 1 0 ... 1 1	1 0 ... 1	0 1 ... 1	...	0 1	0
.....	.....	.....	.....	.....	.....	.....
2^n-2	1 1 1 1 ... 0 1	1 1 ... 0	1 1 ... 0	...	1 0	0
2^n-1	1 1 1 1 ... 1 0	1 1 ... 0	1 1 ... 0	...	1 0	0
0	1 0 0 0 ... 0 0	1 0 ... 0	1 0 ... 0	...	1 0	1
1	0 1 0 0 ... 0 0	1 0 ... 0	1 0 ... 0	...	1 0	1
2	0 0 1 0 ... 0 0	0 1 ... 0	1 0 ... 0	...	1 0	1
3	0 0 0 1 ... 0 0	0 1 ... 0	1 0 ... 0	...	1 0	1
.....	.....	.....	.....	.....	.....	.....
2^n-2	0 0 0 0 ... 1 0	0 0 ... 1	0 0 ... 1	...	0 1	1
2^n-1	0 0 0 0 ... 0 1	0 0 ... 1	0 0 ... 1	...	0 1	1

Table 3

x	$c_0^1c_1^1\dots c_{2^{n-1}-2}^1$	$c_{2^{n-1}-1}^1$	$c_0^2\dots c_{2^{n-2}-1}^2$	…	c_0^{n-1}	c_1^{n-1}	c_0^n
0 … 2^n-1	$0_f0_f\dots0_f$	0_f	$0_f\dots0_f$	…	0_f	0_f	0_f
0 … 2^n-1	$1_f1_f\dots1_f$	1_f	$1_f\dots1_f$	…	1_f	1_f	1_f
0 1	$0_t1_f\dots1_f$	1_f	$0_t\dots1_f$		0_t	1_f	0_t
2 3	$1_f0_t\dots1_f$	1_f					
…	…		…	…			
2^n-4 2^n-3	$1_f1_f\dots0_t$	1_f	$1_f\dots0_t$		1_f	0_t	
2^n-2 2^n-1	$1_f1_f\dots1_f$	0_t					
0 1	$1_t0_f\dots0_f$	0_f	$1_t\dots0_f$		1_t	0_f	1_t
2 3	$0_f1_t\dots0_f$	0_f					
…	…		…	…			
2^n-4 2^n-3	$0_f0_f\dots1_t$	0_f	$0_f\dots1_t$		0_f	1_t	
2^n-2 2^n-1	$0_f0_f\dots0_f$	1_t					

Table 4

X	x_2x_1	$c_0c_1c_2c_3$	$c_0^1c_1^1$	$c_0^2\equiv F$
0	0 0	0 0 0 0	0 0	0
1	0 1	0 0 0 0	0 0	0
2	1 0	0 0 0 0	0 0	0
3	1 1	0 0 0 0	0 0	0
0	0 0	1 1 1 1	1 1	1
1	0 1	1 1 1 1	1 1	1
2	1 0	1 1 1 1	1 1	1
3	1 1	1 1 1 1	1 1	1
0	0 0	0 1 1 1	0 1	0
1	0 1	1 0 1 1	0 1	0
2	1 0	1 1 0 1	1 0	0
3	1 1	1 1 1 0	1 0	0
0	0 0	1 0 0 0	1 0	1
1	0 1	0 1 0 0	1 0	1
2	1 0	0 0 1 0	0 1	1
3	1 1	0 0 0 1	0 1	1

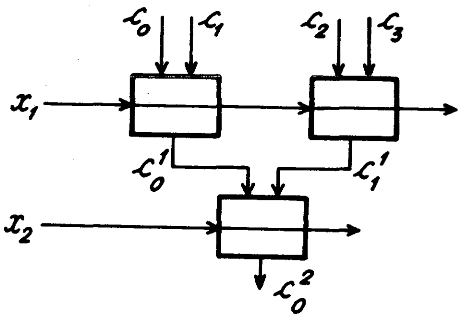


Figure 3—A two level t.c.

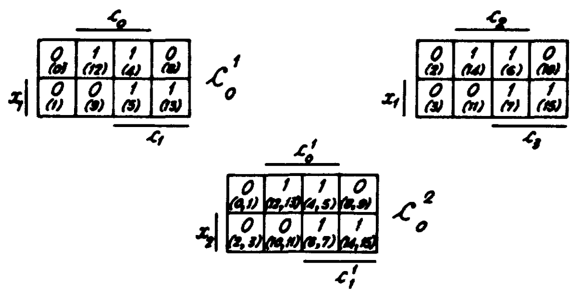


Figure 4—Correspondence between the tests and the entries of the cell maps

controlled in the following order (the columns are numbered from left to right):

(1) the squares of the first columns, first for $x_i = 0$, then for $x_i = 1$, starting from the map of c_0^i as far as the map of $c_{2^{n-i}-1}^i$ (0_f test);

(2) the squares of the third columns, as at point (1) (1_f test);

(3) the upper squares of the fourth columns and then the lower squares of the second columns, as at point (1) (0_t test);

(4) the upper squares of the second columns and then the lower squares of the fourth columns, as at point (1) (1_t test).

Every cell on level i is tested 2^{i-1} times. It is useful for what follows to note the law for the construction of the map of a cell of level i from the maps of the two cells above of level i − 1 (Figure 5).

Properties of the tests

A tree A_n can be decomposed into two subtrees having n − 1 levels, A_{n-1} and A'_{n-1}, and in its output cell. For the sake of simplicity the outputs c_0^{n-1} and c_1^{n-1} of the two subtrees A_{n-1} and A'_{n-1} will be indicated in this section with α and β.

Lemma: If the 0_f, 1_f, 0_t, 1_t tests, carried out on an A_n, give correct outputs, outputs α and β of subtrees

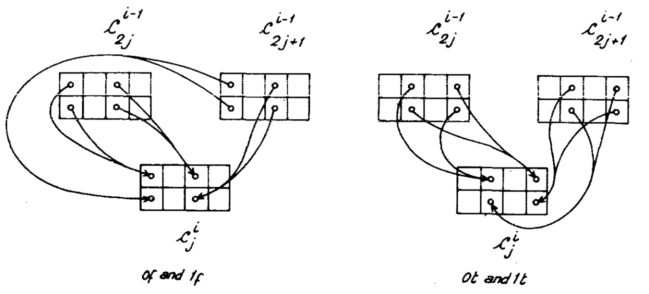


Figure 5—Construction of the c^i_j map from the maps of c^{i-1}_{2j} and c^{i-1}_{2j+1}

A_{n-1} and A'_{n-1} are constant during each half of the four tests.

Proof: Let us suppose that in the 0_f test outputs α and β are not constant, and that the pair of values $\alpha_0\beta_0$ occurs 2k times ($2k < 2^n$), that is k times for $x_n = 0$ and k times for $x_n = 1$ (α and β are independent of x_n). The output of A_n is a function of the specialization bits; in fact, if vector C is changed from (0, . . ., 0) to (1, . . ., 1), the output changes independently of vector X. The same can be said both for α and for β. Therefore in the 1_f test, for the same 2k values of X, α and β must take on values α_1 and β_1. Furthermore $\alpha_0\beta_0$ and $\alpha_1\beta_1$ give the values 0 and 1 to the circuit output independently of x_n and therefore cannot occur in the 1_f and 0_f tests respectively for any of the remaining $2^n - 2k$ values of X. Let $\alpha_1\beta_0$ be another pair of values taken on by α and β during the 0_f test. For the reasons explained previously $\alpha_0\beta_1$ will appear in the 1_f test and cannot appear in the 0_f test; similarly $\alpha_1\beta_0$ cannot appear in the 1_f test. The following table can therefore be constructed (Table 5).

It can be seen that for $x_n = 0$, during the 0_t test, β is bound to take on value β_1, but it is not possible to assign α in such a way as to get 0 in the output: in fact, both $\alpha_0\beta_1$ and $\alpha_1\beta_1$ give 1 in the output independently of x_n. It can be deduced that in the 0_f test, and therefore in the 1_f one, α and β must be constant.

Let $\alpha_0\beta_0$ and $\alpha_1\beta_1$ be the pairs of values taken on by α and β during the 0_f and 1_f tests; Table 6 can be constructed.

Let us consider the 0_t test for $x_n = 0$: β is bound to take on value β_1, consequently $\alpha = \alpha_0$ considering that for $\alpha_1\beta_1$ the output of A_n is 1. In the same way the remaining rows can be constructed, and it can therefore be affirmed that during each half of the four tests α and β remain constant.

Theorem 1: An n-level tree circuit, A_n, functions correctly if and only if tests 0_f, 1_f, 0_t, 1_t give the correct outputs, that is four sequences of zeroes, ones, zeroes, ones respectively, each of 2^n bits.

Table 5

x_n	A_{n-1}	A'_{n-1}	A_n
0,1	0_f: α_0 α_1	0_f: β_0 β_0	0_f: 0
0,1	1_f: α_1 α_0	1_f: β_1 β_1	1_f: 1
0	0_t: ?	1_f: β_1	0_t: 0

Table 6

x_n	A_{n-1}	A'_{n-1}	A_n
0,1	0_f: α_0	0_f: β_0	0_f: 0
0,1	1_f: α_1	1_f: β_1	1_f: 1
0	0_t: α_0	1_f: β_1	0_t: 0
1	1_f: α_1	0_t: β_0	
0	1_t: α_1	0_f: β_0	1_t: 1
1	0_f: α_0	1_t: β_1	

Proof Necessity: it is obvious. Sufficiency: according to the Lemma, if the outputs are correct α and β are constant in each of the eight half-tests. A Karnaugh map can therefore be constructed for the function of the output cell, as shown in Figure 6. The function implemented by this cell is therefore of the following type:

$$F = \alpha^* \bar{x}_n + \beta^* x_n \,. \tag{1}$$

Since the outputs are correct, two possibilities may occur:

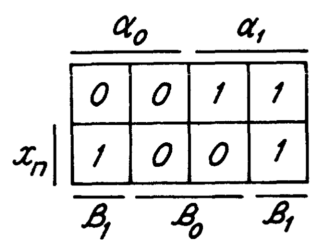


Figure 6—The final cell map

(1) $\alpha^* = \alpha$ and $\beta^* = \beta$: A_{n-1}, A'_{n-1} and the output cell are functioning correctly;

(2) $\alpha^* = \bar{\alpha}$ a/o $\beta^* = \bar{\beta}$: A_{n-1} a/o A'_{n-1} give the complemented output, but the output cell makes up for this defect.

In both cases for $x_n = 0$ {$x_n = 1$} the output of A_n reproduces that of A_{n-1} {A'_{n-1}}, apart from complementations. Therefore the maps of c_0^{n-1} and of c_1^{n-1} can be constructed for $x_n = 0$ and for $x_n = 1$ respectively. It is seen at once that also the functions implemented by these two cells are of type (1), and therefore the considerations made for c_0^n are still valid. Proceeding in the same way, we can construct the maps for all the cells of the circuit, which always implement functions of type (1). It can be concluded that circuit A_n functions correctly, since it has been proved that all the cells function correctly, apart from pairs of faults that cancel each other out and therefore cannot influence the overall behaviour of the circuit.

Theorem 2: The set of tests $0_f, 1_f, 0_t, 1_t$ contains the minimal number of tests necessary to control the functioning of a circuit A_n under the conditions envisaged for the faults.

Proof: In view of the structure of A_n and the type of faults considered, the cells on the first level are all independent of one another; it is evident, therefore, that the number of tests necessary to control them is $8 \cdot 2^{n-1} = 2^{n+2}$. This is obviously a lower limit for the number of tests referring to the whole circuit. Then, too, considering the independence of the cells of each level, the attempt can be made to organize the 2^{n+2} tests necessary to control the cells of the first level so as to carry out at the same time the tests on all the cells of the following levels. Theorem 1 shows that this is actually possible. This is what we set out to prove.

The utilization of faulty cellular tree circuits

Fault localization

The method set forth in the preceding section also makes it possible to localize the faulty cells in a way that will be described. As has been seen, tests 0_f, 1_f, 0_t, 1_t make it possible to construct the Karnaugh map for the cell at level n, and every square of the map is explored by 2^{n-1} tests. At level i a cell c_k^i is tested 2^{i-1} times. If this cell is faulty, exhibiting a mistake in a square corresponding to $x_i = 0$ {$x_i = 1$}, 2^{i-1} consecutive wrong outputs occur starting from element $2k \cdot 2^{i-1}$ {$(2k + 1) \cdot 2^{i-1}$} of one of the four sequences.

The same output sequence is obtained if the two cells c_{2k}^{i-1} and c_{2k+1}^{i-1} are faulty in the same way as c_k^i. Similarly it can be seen that 2^h cells at level $i-h$ with faults in the same squares of the maps (and all the combinations of these faults) lead to the same output sequence.

It can be concluded, therefore, that, if in one or more tests there exist as many output sequences composed of 2^{i-1} wrong terms starting from element $2k \cdot 2^{i-1}$ or $(2k + 1) \cdot 2^{i-1}$, the subtree having c_k^i as its output cell contains one or more faulty cells. If there exist several wrong sequences not correlated in the preceding way, there exist as many faulty subtrees in the circuit.

The data supplied by the four tests do not permit a more exact localization of the faults. Then, too, it is easily seen that the number of additional tests necessary for this purpose can become prohibitive in proportion as the dimensions of the faulty subtree A_i grow. Furthermore, a minimal set of these tests cannot generally be fixed in advance independently of the results of the preceding tests.

A method will be described which makes it possible, on the contrary, to use a faulty A_n to implement a subset of functions of n variables without further diagnostic tests.

Finding a set of functions that can be implemented by a faulty t.c.

Let us suppose that the tests 0_f, 1_f, 0_t, 1_t have detected a subtree A_i containing one or more faulty cells. Cases $i = 1$ and $i > 1$ are considered separately.

1st case ($i = 1$): Tree A_i is reduced to only one cell of the first level. Since the inputs of this cell are all accessible from the outside and therefore known, it is easy to obtain the function implemented by the faulty cell. Let this function be $c_j^1 = g(x_1, c_{2j}, c_{2j+1})$. If it is expanded according to Shannon with respect to x_1, we get

$$g = \bar{x}_1 \, g_0(c_{2j}, c_{2j+1}) + x_1 \, g_1(c_{2j}, c_{2j+1}). \qquad (2)$$

Terms g_0 and g_1 can be considered as equivalent specialization bits and they depend generally on both the signals applied to the two control terminals of the cell.

If and only if the correspondence between pairs $g_0 g_1$

and $c_{2j}c_{2j+1}$ is a permutation, cell c_j^1 is still able to implement all the functions of x_1, and therefore A_n is able to implement all the functions of n variables. In all other cases the functions of x_1 that can be implemented by c_j^1 and therefore the functions of n variables that can be implemented by A_n can be found immediately.

Example 1: Let us consider an A_3 (Figure 7). The result of the diagnostic tests is supposed to be the one shown in Table 7.

In each test one wrong isolated term is noted; therefore the faults of the circuit are limited to the first level. Since the wrong terms correspond to $X = 2$ and $X = 3$, it can be deduced ($k = 1$) that they all correspond to cell c_1^1; the map of the function implemented by this cell is therefore that shown in Figure 8. The function implemented by c_1^1 in form (2) is therefore

$$c_1^1 = \bar{x}_1\, \overline{c_2c_3} + x_1\, c_2c_3\, .$$

The equivalent specialization bits are:

$$c_{2eq} = \overline{c_2c_3} \quad ; \quad c_{3eq} = c_2c_3\, .$$

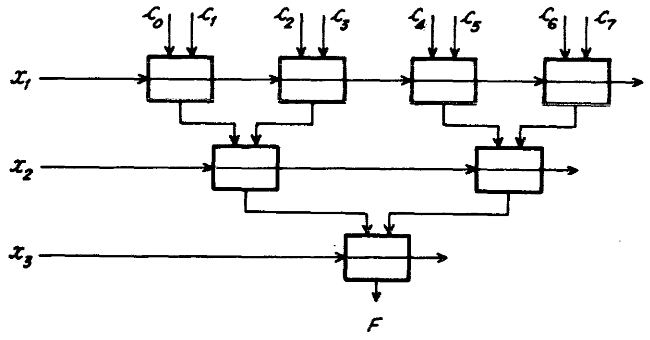


Figure 7—The t.c. considered in example 1

Table 7

X	0_f	1_f	0_t	1_t
0	0	1	0	1
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	0	1	0	1
5	0	1	0	1
6	0	1	0	1
7	0	1	0	1

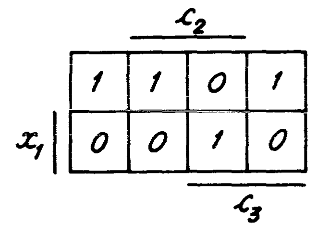


Figure 8—The map of the faulty cell of example 1

Owing to the correspondence between the specialization bits and the minterms of the functions implemented by A_3, it can be deduced that all the functions having one and only one of minterms m_2 and m_3 can be implemented. To implement a function with m_2 alone the terminals can be specialized in the usual way; to implement a function with m_3 alone it is necessary, on the contrary, to fix c_2 and c_3 both at 1.

2nd case ($i > 1$): Two types of faults can be distinguished as can be seen from the map of cell c_j^i which has

(a) one correct row.
(b) errors in both rows.

(a) If cell c_j^i has the map with the row $x_i = 0$ $\{x_i = 1\}$ correct, it can be said that subtree A_i functions correctly when $x_i = 0$ $\{x_i = 1\}$ according to the considerations set forth in Theorem 1. The faulty tree can therefore be decomposed as in Figure 9.

When $x_i = 0$ $\{x_i = 1\}$ the output c_j^i coincides with the output of A_{i-1} $\{A'_{i-1}\}$. Then, too, when $x_i = 1$ $\{x_i = 0\}$ the output of c_j^i can be known independently of $x_1, \ldots, x_{i-1}$ if all the specialization bits of A'_{i-1} $\{A_{i-1}\}$ are fixed at 0 or at 1. This output coincides with the value of the lower $\{$upper$\}$ square corresponding to test 0_f or 1_f on c_j^i.

An A_n with such faults can implement functions of the following type

$$F(X) = X^j_{n-i}\,[x_i^{*}\, f(x_1, \ldots, x_{i-1}) + \bar{x}_i^{*}\, \gamma] + \\ + \sum_{h\neq j} X^h_{n-i}\, g_h(x_1, \ldots, x_i) \tag{3}$$

where: $x_i^{*} = \bar{x}_i$ $\{x_i\}$ if the upper $\{$lower$\}$ half-map is correct;

X^j_{n-i} is the vector $(x_{i+1}, \ldots, x_n)$ direct or complemented so that $c_0^n = c_j^i$ for $X^j_{n-i} = 1$.

γ is defined in Figure. 10 for $x_i^{*} = x_i$; for $x_i^{*} = \bar{x}_i$, γ is defined in a similar way, taking into account the lower half-map; σ_{i-1} is the value common to the 2^{i-1} specialization bits of A'_{i-1} $\{A_{i-1}\}$;

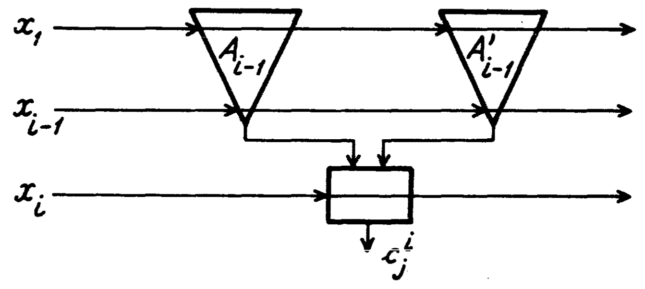


Figure 9—Decomposition of a faulty tree A_i

f and g are any functions of their arguments. They are therefore programmed on their subtrees specializing the bits in the usual way.

If there are r subtrees $A_{i_1}, \ldots, A_{i_r}$ which show faults of this type, the proceeding is similar. The circuit can implement functions of the type:

$$F(X) = \sum_{q=1}^{r} X_{n-i_q}^{j_q} [x_{i_q}^{*} f_q(x_1, \ldots, x_{i_q-1}) + \bar{x}_{i_q}^{*} \gamma_q] + \tag{4}$$

$$+ \prod_{q=1}^{r} \sum_{h_q \neq j_q} X_{n-i_q}^{h_q} g_{h_q}(x_1, \ldots, x_{i_q}) \; .$$

In fact, owing to the structure of the tree, the first terms of Equation 3 written for the individual faults can be implemented independently. The second terms of Equation 3 must be intersected with one another so as to obtain the residual part of A_n which does not belong to any of the faulty subtrees.

Example 2: Given an A_4, it is supposed that the result of the tests is as shown in Table 8.

There is a sequence 0000 in column 1_t for X = 0, 1, 2, 3; this implies the presence of a fault in a subtree A_3 ($2^{i-1} = 2^2$), which has c_0^3 as its output cell (k = 0). Similarly, the two sequences 00 in tests 1_f and 1_t corresponding to X = 10, 11 imply the presence of a fault in an A_2 with c_2^2 as its output cell. The maps corresponding to c_0^3 and c_2^2 are shown in Figure 11. Both have one correct row. Applying Equation 4 we obtain:

Table 8

X	0_f	1_f	0_t	1_t
0	0	1	0	0
1	0	1	0	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	1
5	0	1	0	1
6	0	1	0	1
7	0	1	0	1
8	0	1	0	1
9	0	1	0	1
10	0	0	0	0
11	0	0	0	0
12	0	1	0	1
13	0	1	0	1
14	0	1	0	1
15	0	1	0	1

$$F(X) = \bar{x}_4 x_3 f_1(x_1, x_2) + \bar{x}_3 \sigma_3 + x_4 \bar{x}_3 \bar{x}_2 f_2(x_1) + x_2 \cdot 0 +$$
$$+ x_4 g_1(x_1, x_2, x_3) \bar{x}_3 \bar{x}_4 g_{2,1}(x_1, x_2) +$$
$$+ x_3 \bar{x}_4 g_{2,2}(x_1, x_2) + x_3 x_4 g_{2,3}(x_1, x_2) =$$
$$= x_3 \bar{x}_4 f_1(x_1, x_2) + \bar{x}_3 \bar{x}_4 \sigma_3 + \bar{x}_2 \bar{x}_3 x_4 f_2(x_1) +$$
$$+ x_3 x_4 f(x_1, x_2) \; .$$

In the last term $f(x_1, x_2)$ has been used instead of $g_1(x_1, x_2, 1)\, g_{2,3}(x_1, x_2)$ since this product is any function of x_1 and x_2.

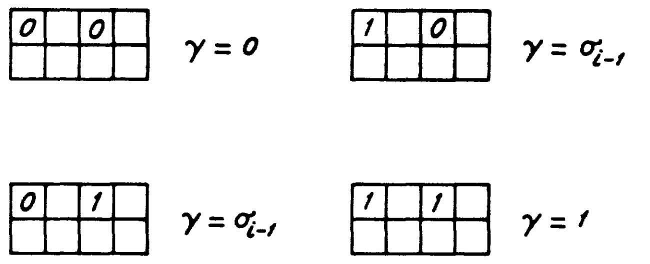


Figure 10—Definition of the function γ

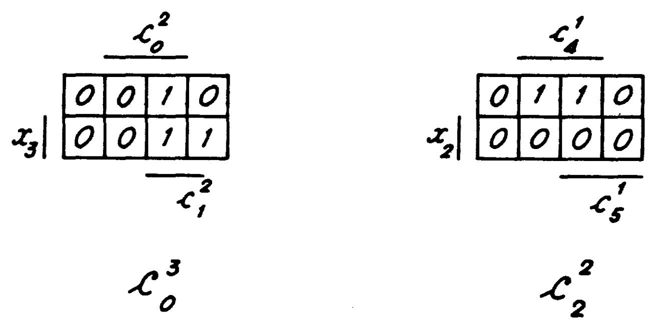


Figure 11—The maps of the faulty cells of example 2

(b) If the map of c_j^i does not have a correct row, let us consider the functioning of A_i in tests 0_f and 1_f. Since every square of the map of c_j^i is explored 2^{i-1} times corresponding to all the values assumed by $x_1, \ldots, x_{i-1}$, the output of A_i in these tests is a function of x_i alone or is constant, according as to whether the columns of the map of c_j^i corresponding to tests 0_f and 1_f are 01, 10 or 00, 11. Therefore A_n can implement functions of the type:

$$F(X) = X_{n-i}^{j}\,[\sigma_i c_j^i(x_i)_1 + \bar{\sigma}_i c_j^i(x_i)_0] + + \sum_{h \neq j} X_{n-i}^{h}\, g_h(x_1, \ldots, x_i) \tag{5}$$

where: σ_i is the value common to the specialization bits of A_i;

$c_j^i(x_i)_1$ and $c_j^i(x_i)_0$ are the functions implemented by cell c_j^i during the tests 1_f and 0_f respectively;

the g_h are any functions of $x_1, \ldots, x_i$ and are programmed on their subtrees by specializing the bits in the usual way.

In the case of r subtrees $A_{i_1}, \ldots, A_{i_r}$ with faults of this type, we obtain:

$$F(X) = \sum_{q=1}^{r} X_{n-i_q}^{j_q}\,[\sigma_{i_q} c_{j_q}^{i_q}(x_{i_q})_1 + \bar{\sigma}_{i_q} c_{j_q}^{i_q}(x_{i_q})_0] + + \prod_{q=1}^{r} \sum_{h_q \neq j_q} X_{n-i_q}^{h_q}\, g_{h_q}(x_1, \ldots, x_{i_q}) \,. \tag{6}$$

Example 3: Let the results of the tests carried out on an A_4 be those shown in Table 9. The presence of two faulty subtrees with output cells c_0^3 and c_2^2 can be noted. Their respective maps are shown in Figure 12.

If we apply Equation 6, we get:

$$\begin{aligned} F(X) &= \bar{x}_4(\sigma_3 x_3 + \bar{\sigma}_3 x_3) + \bar{x}_3 x_4(\bar{\sigma}_2 \bar{x}_2 + \sigma_2 \cdot 0) + \\ &+ [x_4 g_1(x_1, x_2, x_3)] \cdot [\bar{x}_3 x_4 g_{2,1}(x_1, x_2) + \\ &+ x_3 \bar{x}_4 g_{2,2}(x_1, x_2) + x_3 x_4 g_{2,3}(x_1, x_2)] = \\ &= x_3 \bar{x}_4 + \sigma_2 \bar{x}_2 \bar{x}_3 x_4 + x_3 x_4 f(x_1, x_2) \,. \end{aligned}$$

The number of the functions that can be implemented by a faulty circuit according to Equations 4 and 6 can be considerably increased if we admit the possibility of permutating the variables $x_1, \ldots, x_n$ on the n inputs. To be exact, the number of functions that can be implemented should be multiplied by n!. In the case of Example 2, for instance, the functions

Table 9

X	0_f	1_f	0_t	1_t
0	0	0	0	1
1	0	0	0	1
2	0	0	0	1
3	0	0	0	1
4	1	1	1	0
5	1	1	1	0
6	1	1	1	0
7	1	1	1	0
8	0	1	1	1
9	0	1	1	1
10	0	0	0	1
11	0	0	0	1
12	0	1	0	1
13	0	1	0	1
14	0	1	0	1
15	0	1	0	1

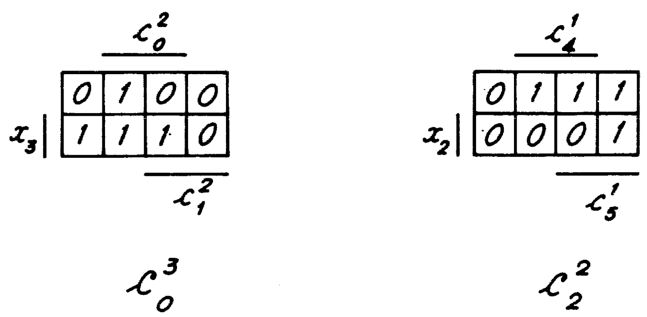


Figure 12—The maps of the faulty cells of example 3

implemented are $2^{11} = 2048$; by permutating the variables in any way, they become $2^{11} \cdot 4! = 49152$.

CONCLUSIONS

A diagnostic method for t.c. has been described which makes it possiblè to utilize faulty circuits in a simple way. The diagnostic tests for the circuit are in minimal number and can easily be carried out automatically, in view of their independence of the faults and their uniformity. In the use of faulty t.c. one or more subtrees containing the faulty cells are isolated and their specialization bits are all fixed at 0 or at 1. In this way it is not possible to find all the functions that the faulty t.c. is still able to implement. However a large number of additional tests, which furthermore cannot be established in advance, would be necessary to determine the complete set of functions that can be implemented on a faulty A_n. Then, too, the law for giving the specialization terminals their proper values would be generally somewhat complicated.

REFERENCES

1 R C MINNICK
A survey of microcellular research
Journal of the A C M vol 14 no 2 April 1967 pp 203–241
2 R C MINNICK
Cutpoint cellular logic
IEEE Trans vol EC–13 no 6 Dec 1964 pp 685–698
3 G CIOFFI V FALZONE
Circuiti cellulari con struttura ad albero
Automazione e Strumentazione vol 16 no 8 Aug 1968 pp 338–350
4 S S YAU C K TANG
Universal logic circuits and their modular realization
Proc S J C C 1968

Solid state keyboard

by EVERETT A. VORTHMANN

Honeywell, Inc.
Freeport, Illinois

and

JOSEPH T. MAUPIN

Honeywell, Inc.
Plymouth, Minnesota

INTRODUCTION

The computer industry is no doubt one of the most rapidly growing industries today. With the increase in computer usage, there is an increased demand to improve man's ability to communicate with the computer. The prime instrument of input communication today is a keyboard, and it appears that this will be true for some time in the future.

The requirements of today's keyboards are becoming more complex. Increased reliability and more flexibility to meet specialized demands are essential. Remote terminals are quite often operated by relatively untrained personnel and the keyboard must be capable of error-free operation for these people. At the same time it should be capable of high thru-put for the trained operator as will be used on a key tape machine.

Some of the limitations of existing keyboards are:

- Mechanical interlocks which reduce operator speed.
- Excessive service (increasingly important for remote terminals).
- Contact bounce and wear of mechanical switches.
- Non-flexible format.

Our general objective in developing the solid state keyboard was to overcome these limitations and still have a competitively priced design.

Keyboard organization

Before setting forth specific design objectives, some general comments may be helpful, depending on the reader's familiarity with this type of equipment. The purpose of a keyboard is to feed information into a digital computer by means of a binary code. An eight bit parallel code is usually used. Two codes currently used are ASCII and EBCDIC and the keyboard design and construction must be such that any code can conveniently and economically be supplied. The output signal must be compatible with the solid state integrated circuits used in today's computers.

Specific design objectives

At this point, each key may be thought of as a simple switch, actuated by the position of a key plunger.

Human factors studies[1-4] helped establish the following for mechanical and tactile features of key operation.

Operate force: 2 to 5 ounces

Pretravel (before operate): .075 inches min. from free position

Release point: .040 inches min. from free position

Release point (R.P.) $\leqq$ Operate point (O.P.) $- \delta$ where $\delta =$ differential

Switching transitions should be "snap acting" or regenerative so that it will not be possible to hold a key in a position that will cause ambiguity at the output. Rise and fall times must be in the low microsecond range without any ringing or oscillation. The encoding electronics must be capable of blocking error codes when two or more keys are depressed.

Keyboard formats are quite varied, depending on the user's needs and preferences. This indicated that each key should be a separate module. Finally, the service life should be in excess of 10 million operations per key, and low cost was given a priority second only to reliability.

The approach

From the outset, our thinking was slanted toward the development of an integrated circuit chip transducer for the key module. The powerful and still growing economic advantages of batch processing used in integrated circuit manufacture were considered essential to our stringent cost objectives. To fully exploit these advantages, it was desirable that the chip be complete in itself i.e., that it require no external components to accomplish its function. The latter cannot be added in batch fashion.

Several approaches to mechanical position detection without contacts were studied, based mostly on unique sensor effects available in a semiconductor such as silicon. Position control of an electric, magnetic, acoustic, or electromagnetic (including optical) field pattern is fundamentally involved. Hall effect sensing in a silicon device, including appropriate integrated electronics, and coupled with permanent magnet actuation, was singled out for detailed analysis, design and development.

This approach was eventually adopted for the solid state keyboard. The competing approaches mentioned above, though quite feasible, seemed to require more expensive packaging, or more expensive and less reliable field sources, or were known to require external components in the electronics. Magnetic actuation looked particularly attractive because it appeared that the sensor device could be integrated on the silicon chip along with associated electronics as well as allowing more freedom in package selection. Among the several galvanomagnetic effects present in semiconductors, we found that only the Hall effect in silicon is large enough to be useful in the low field (< 1000 Gauss) region of interest.

The silicon chip that has been developed is described by the functional breakdown in Figure 1.

A circuit schematic is given in Figure 2, and a picture of the chip comprises Figure 3.

Some of the analytical and experimental investigations associated with this development are presented in what follows.

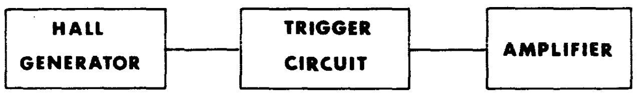


Figure 1—Functional description of silicon chip transducer

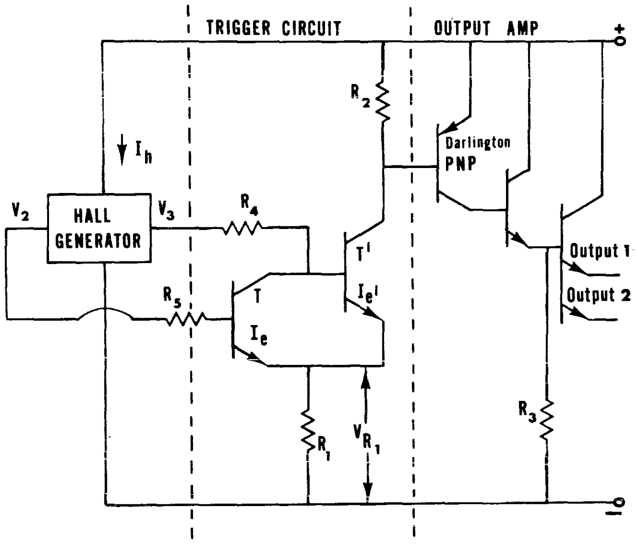


Figure 2—Circuit schematic of chip transducer

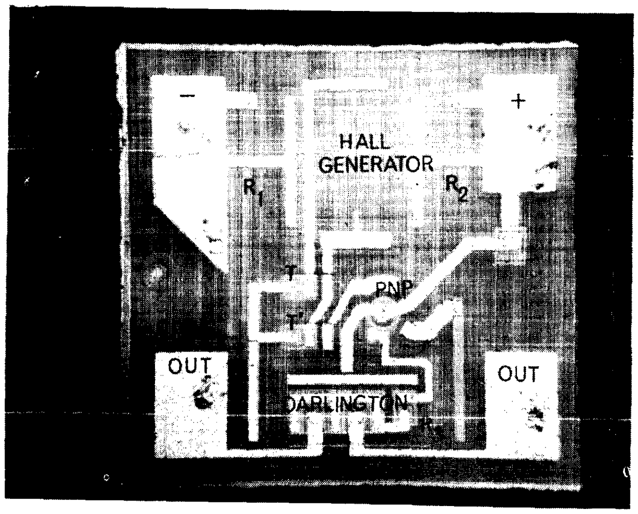


Figure 3—Photograph of chip

Hall effect characterization

As many readers will already know, the Hall effect is old in scientific terms, having been discovered by Edward W. Hall in 1878. It is currently enjoying a renaissance of practical applications interest due mainly to advances in semiconductor technology. Two publications[5,6] by A. C. Beer provide excellent general references. The Hall effect results directly from the Lorentz force on moving charge carriers, where the average motion is constrained in direction, as in a solid. This is illustrated in Figure 4. The Lorentz force creates a charge unbalance in the y-direction. The resulting

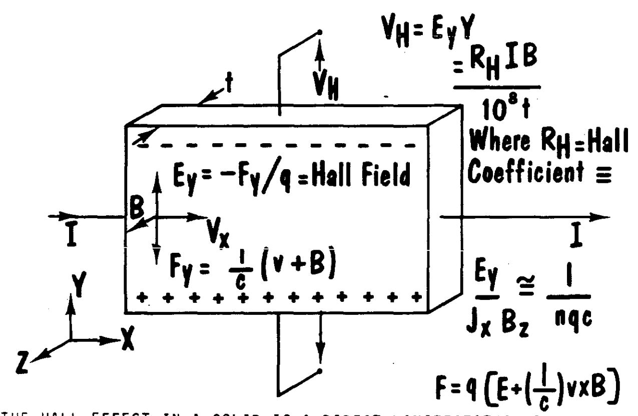

$V_H = E_Y Y = \frac{R_H I B}{10^8 t}$
Where R_H = Hall Coefficient ≡
$\frac{E_Y}{J_X B_Z} \cong \frac{1}{nqc}$
$F = q\left(E + \left(\frac{1}{c}\right) v \times B\right)$
$E_Y = -F_Y/q$ = Hall Field
$F_Y = \frac{1}{c}(v+B)$
V_X
V_H
I
B
t
Y
X
Z
THE HALL EFFECT IN A SOLID IS A DIRECT MANIFESTATION OF THE LORENTZ FORCE ON MOVING CHARGE CARRIERS:

Figure 4—Hall effect illustration

electric field, called the Hall field, provides a force that cancels the Lorentz force. Integration of the Hall field between a pair of probe contacts on the sides of the conductor produces the Hall voltage.

Equation 1 shows an approximate Hall voltage expression for a homogeneous layer with predominately one type of carrier concentration.

$$V_H = \frac{R_H \, IB}{10^8 \, t} \qquad (1)$$

R_H	= Hall* coefficient $\simeq \frac{1}{nq}$
n	= concentration (cm^{-3})
I, B, V_H	= mutually perpendicular
t	= Thickness (cm)
B	= Gauss
V_H	= volts
I	= amperes
q	= charge on an electron

Since most electronic circuits operate from a constant voltage supply, equation 2 below is more appropriate, It is straight-forward derivation* from equation 1 for a rectangular geometry.

* We assume "Hall mobility" and "conductivity mobility" to be substantially equal for the conditions of interest. The validity of this assumption, has been confirmed through private communication with Dr. G. D. Long, Honeywell Corporate Research Center.

$$V_H = \frac{V_s u_d B \, W \, 10^{-8}}{L} \qquad (2)$$

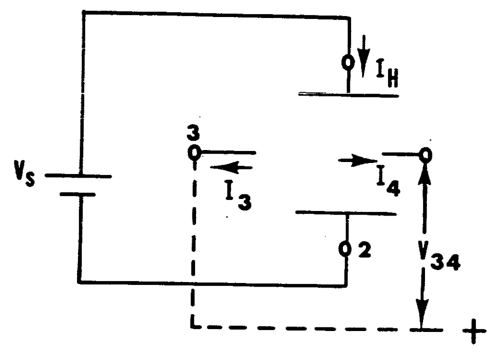
V_s	= supply voltage
u_d	= mobility (cm^2/v-sec)
W	= width
L	= length

A factor less than unity has to be applied to equation 2 if the aspect ratio W/L is not smaller than unity. This is due to the shorting effects of the end contacts on the Hall field. One cannot increase the Hall voltage indefinitely by increasing W/L.

Equation 2 illustrates the important role of carrier mobility. In this respect, silicon is not a good material[5] compared to, say, InSb or GaAs. However, one must go beyond equations 1 and 2 to include practical constraints of power dissipation, electrical resistance, range of impurity concentrations, and temperature variation of Hall coefficient. When this is done, silicon looks much better. Relating constant current and constant voltage modes of operation to semiconductor processing, observe that thickness and concentration (equation 1) are also the major processing variables that control resistor values in integrated circuits, and the expected tolerance is quite large. On the other hand, with constant voltage, only the mobility is process dependent, and it tends to be a weak function[7] of concentration in the region of interest. Thus we expect and obtain much better reproducibility of Hall voltage with constant voltage excitation. This is gained at the expense of a higher temperature coefficient, however, due to the variation of mobility with temperature.

Assuming a field of 1000 Gauss to be available, straight-forward calculations using typical mobility and reasonable geometries showed that we could obtain a signal of about 30 millivolts with a five volt supply, and without exceeding typical power dissipation capabilities in IC chips.

Figure 5 shows an expression for total d.c. output voltage of a Hall element, including the effects of small loading at the Hall terminals. The characterization is entirely in terms of parameters measurable at the terminals. The offset voltage term, Vq, which is the open circuit output voltage with zero magnetization, is a very important parameter in this device. Economic restrictions ruled out the use of external resistors for adjustment of Vq. Its nominal value using IC technology depends mostly on contact geometry and sheet resistance uniformity in the conducting layer comprising the Hall element. Fortunately, very accurate geometries are possible using photolithographic techniques developed



If $(I_3 + I_4) \ll I_H$;

$V_{34} = V_q + V_H - I_3R_3 \mp I_4R_4$

V_q = Offset Voltage; V_H = Hall Voltage

Figure 5—Hall element output voltage characterization

for IC fabrication. Variations in Vq can be caused by several factors, such as internal stress (through the piezoresistance effect) and temperature gradients. Regardless of the nature of the electronic circuitry that follows the Hall element, the variations in Vq must be much lower than the Hall voltage for adequate "signal-to-noise" ratio.

Design—Process interrelationships

As with any integrated circuit development, the circuitry, device physics, and process techniques are interdependent, and must be so treated. At the time the development was initiated it was considered essential for low cost objectives to use the epitaxial-diffusion,[9] bipolar, NPN based, type of processing which was rapidly becoming an industry standard. MOS type processing was not sufficiently controllable to be seriously considered.

In an NPN type of bipolar structure, the collector layer has the lowest carrier concentration and highest mobility; it is the best choice for a Hall element. Hence the design approach was pursued on the basis of forming the Hall element simultaneously with collector regions for NPN transistors. The same isolation diffusion is used for defining the Hall element geometry. (The Hall element outline is faintly visible in Figure 3). Since this is a novel type of Hall element structure, some preliminary experimentation was conducted, confirming its feasibility and the accuracy of the preceding characterizations.

As to process considerations for the associated circuitry, the objective was to take advantage of conventional IC processing strengths, which lead to high yield results on the following:

1. High gain, accurately matched NPN transistors.
2. Low gain PNP transistors.
3. Accurate control of resistance ratios, but not absolute values.

Throughout the design-development cycle, extensive effort was devoted to achieving a simple design that is amenable to high yield processing, yet adequate for the intended function without external components.

The trigger circuit

The function of the trigger circuit is to accept the linear output (with or without linear amplification) of the Hall element and convert it to a binary or ON-OFF mode, with regenerative switching transitions and controllable hysteresis (or differential between the "turn on" and "turn off" operate points).

The trigger circuit we devised is shown in Figure 6. It is a variation on the Schmitt type of circuit. It may be implemented with just two resistors and two bipolar transistors. An approximate analysis aimed at providing insight into its general characteristics will be given here.

Assumptions used in approximate analysis:

a. The transistor model shown in Figure 6 applies. The most important feature of this model is that Shockley's law applies to the $I'_E - V_{BE}$ characteristic. This is well established for silicon planar bipolar transistors. Extrinsic resistances, collector conductance and all time dependent effects are omitted. The model requires active region operation, which is easily met.
b. $I_E + I_E' = I$, a constant
c. $I_B \ll I_C$ (high gain)

The static voltage control characteristics at the input base is of primary interest.

$$V_B = V_{BE} - V_{BE}' - I_C R_4 \qquad (3)$$

Using the above assumptions this becomes,

$$V_B = -\frac{KT}{q}\left(\ln\frac{I_s}{I'_s} + \ln\frac{1 - I_E/I}{I_E/I}\right) - (\alpha I R_4)\, I_E/I \qquad (4)$$

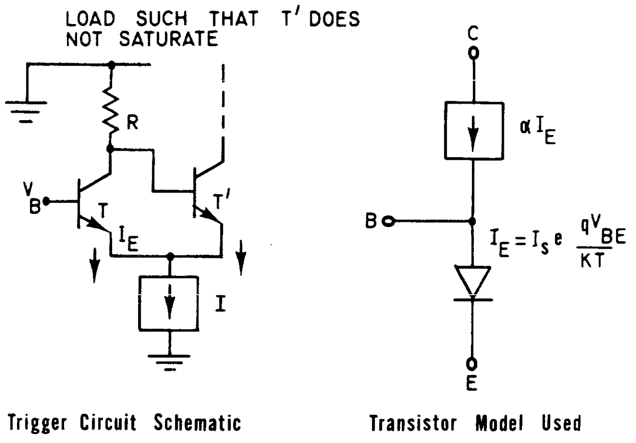


Figure 6—Trigger circuit schematic

The first term of equation 4 represents the control characteristic[10] of a conventional difference amplifier stage using the same assumptions noted above. The second term is the result of linear regenerative feedback. If this term has the appropriate magnitude, the transfer characteristic will have a negative resistance region that covers a few millivolts. The transfer characteristic may be easily observed on a curve tracer using a discrete component version of the circuit. One such observation is reproduced in Figure 7. The constant total emitter current condition is approximated in this version of the circuit by using an emitter resistor with voltage drop that is several times larger than the V_{BE} voltage.

The nature of regenerative switching transitions may be reviewed in a number of references, particularly those dealing with tunnel diode circuits. Chapter 15 in Linvill and Gibbons' book[8] is especially good. We will only note here that the trigger points depend exclusively on static parameters, and are given by the transition points from positive to negative resistance around any closed mesh of the circuit. Reactive effects, active device response time, etc., affect the speed of the

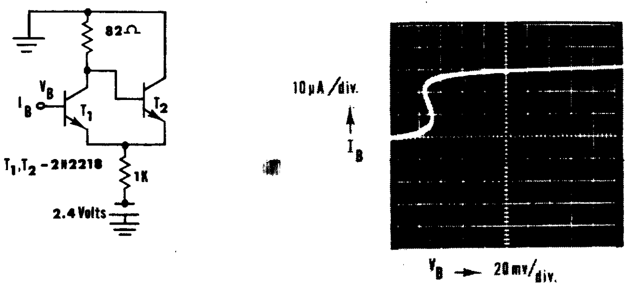


Figure 7—Experimental look at trigger circuit control characteristic

switching transition, but not the fact of its occurrence. The trigger points are thus found by taking the derivative of equation 4, shown below as equation 5, equating it to 0, and simultaneously solving equations 4 and 5.

$$\frac{dV_B}{d\left(\frac{I_B}{I}\right)} = \frac{KT}{q}\,\frac{1}{(1 - I_B/I)\,\frac{I_B}{I}} - \alpha IR_4 = 0 \qquad (5)$$

A result from this approximate analysis is given below for one condition of regenerative feedback. The "ON" condition is defined as T′ conducting and T off.

αIR_4	Turn ON point		Turn OFF point	
	V_B	$I_{B/I}$	V_B	$I_{B/I}$
.130 volts	−.069 volts	0.723	−.061 volts	0.276

$$\frac{KT}{q} = .026 \text{ volts, and } I_s = I'_s$$

Our investigations showed that this circuit configuration could provide regenerative switching transitions with rather precisely defined trigger points and voltage transitions between trigger points of a few millivolts.

The component requirements are well suited for integration, with critical performance depending on transistor matching and resistance ratios. Note that the transistor matching requirements are the same as for a good difference amplifier stage, with V_{BE} matching to about ±2 millivolts. This is routinely done in IC's, due to close physical proximity, extremely accurate matching of geometries, and simultaneous processing.

Output amplifier

The output amplifier, consisting of a PNP stage driving an NPN Darlington, operates in standard saturated switching logic fashion. Its characteristics are relatively non-critical. The PNP is fabricated with a "lateral" geometry and its current gain is low. Static conditions for the OFF and ON states are as follows:

	INPUT	*OUTPUT*
OFF State	T′ Off zero drive to PNP base	Output voltage = 0 (with reference to—supply)
ON State	T′ On PNP saturated	Output voltage = supply voltage minus (V_{CE} SAT$_3$ + V_{BE4} + V_{BE5})

As previously noted, a functional requirement is that there be no linear region in the output of the device, i.e., output voltages between the OFF and ON levels can only exist on a transient basis. This requires that the thresholds associated with the output amplifier operation be well within the negative resistance region of the trigger circuit control characteristic. The resistor R_2 is designed such that the value of $I'C$ at the trigger circuit turn on point will not develop enough voltage across R_2 to forward bias the PNP base-emitter junction. The combined PNP-NPN gain requirement is such that the PNP stage saturates at a value of I'_C that is below the trigger circuit turn off point. Resistor R_3 provides adequate margin against self turn-on in the Darlington stage under worst case temperature and gain conditions.

The output transistor has dual emitters and provides two isolated outputs. This aids in the encoding logic; in effect, part of that logic is included in the chip. This is an example of the economics possible when using IC technology, for the additional output adds virtually no cost to the chip.

An additional benefit of the solid state keyboard is that the output signal from the key does not require additional buffering to eliminate the effects of contact bounce. Switching times are in the low microsecond range and are free from ringing or oscillation.

Integration of sensor and electronics

Aside from the usual considerations of parasitic interactions within an integrated circuit, the special effects resulting from including the sensor in the IC chip constituted an interesting and novel aspect of this development. In general, we find more advantages than disadvantages in this approach and predict a growing trend toward "integrated transducer" semi-conductor devices. Inductance parasitics are virtually eliminated due to the extremely small dimensions. A potential source of ringing or oscillation in regenerative switching circuitry is thus avoided. For the same reason, noise pickup in the leads from sensor to electronics is minimized. High impedance leads to the outside world are avoided.

In the functional operation of this device, magnetization is applied over the entire chip. This has no effect on the electronics, as expected, for the resistors and transistors do not have any magnetic sensitivity in the magnetic field range of interest. The Hall element output, like a balanced bridge with matched temperature sensitive resistors, is sensitive to temperature gradients. This has to be taken into account in the output stage design and operation, and the thermal design of the package. The most troublesome parasitic encountered has been the stress sensitivity of the Hall element through the piezoresistance effect, previously mentioned, and this has been overcome by some special mechanical features in the chip-package design.

Chip specification

The specification is given in Table I. It is written as broad as possible to maximize the overall process yield.

Computer aided analysis

In the design of a product intended for the computer field, the utilization of computer-aided analysis seems especially fitting. When we avoid some of the simplifying assumptions used in the preceding approximate analysis, equations analogous to (4) and (5) become extremely cumbersome. Their simultaneous solution to obtain operate and release points becomes humanly intractable; a computer program was written to obtain such solutions.

Performance of the device was studied as a function of several independent parameters.

1. Supply voltage
2. Transistor gain, matched and prescribed mismatch
3. Emitter junction saturation current, matched and prescribed mismatch
4. Resistor and resistor ratios R_1, R_4/R_1.
5. Offset voltage, Vq

Space does not allow presentation of this analysis and results. The reader may contact the authors if interested. The computer-obtained results have been of great value in guiding the design and the design-process relationship. Figure 8 shows the effect of gain and

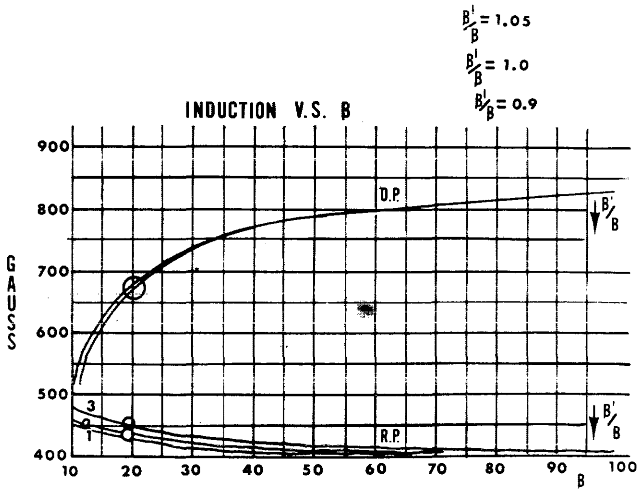


Figure 8—Effect of gain and gain mismatch

gain mismatch. We note that performance becomes essentially independent of transistor beta in the range above 50. With these results, a realistic process gain specification minimum of 30 was established.

Temperature characteristics

The dependence of operate and release points on temperature for a typical device is shown in Figure 9, based on experimental data. The slope of the curves is roughly accounted for by the expected temperature dependence of mobility. However, second order effects in the circuitry have a certain influence, not completely analyzed at this time. First order temperature effects in the circuitry are eliminated by use of matching and ratioing techniques.

Packaging the chip

Upon examining the economics of integrated circuits, it becomes apparent that much of the cost of commercial integrated circuits is in chip packaging rather than the chip itself. It was necessary, therefore, to develop a low cost, reliable packaging technique suitable for magnetic operation.

In developing such a package there are many parameter trade-offs that must be made in order to arrive at an optimum configuration. In most standard chip packaging approaches the chip is eutecticly bonded to a metal leadframe or header. The metal is normally kovar which closely approximates the thermal expansion of the silicon chip. Since this device was to be magnetically operated, kovar is not desirable because it is ferromagnetic. On examining the non-magnetic metals and alloys, it was evident that there were none with the proper thermal coefficient of expansion. Therefore, it was necessary to find another method of holding the chip. The approach selected was to allow the chip, in essence, to float in a non-rigid potting material. This is accomplished in the following manner: A leadframe is stamped from phosphor bronze, inserted into a mold, and transfer molded with a rigid plastic leaving a cavity for the chip and access to the ends of the leadframe as is shown in Figure 10. It should be noted that the cavity for the chip is entirely plastic.

The chip is inserted into the cavity and the four wires are ultrasonically bonded between the pads on the chip and the leadframe. At this point, the chip is held in place by the four wires. The final packaging operation is to fill the cavity with a silicone potting material, which has a very low viscosity in the uncured condition, and it completely encapsulates the chip including the reverse side. Figure 11 shows the chip in its cavity prior to being potted.

In order to minimize the cost of this packaging approach, it was necessary to design so that wire bonding could be automated. This was accomplished in the following manner. The wafer is sawed into chips with an abrasive slurry, rather than use the normal scribe and break process. The sawing produces chips with square edges and with dimensions controlled to within ± .001 inches. The chip cavity is made only slightly larger than the maximum chip size; hence the location of the pads on the chip relative to the leadframe is rather precisely controlled. This allows the wire bonding machine to be mechanically aligned, rather than require the operator to make a visual alignment for each bond. It should also be noted from Figure 3 that the pads on the chip are large—(by integrated circuit standards)—approximately .010″ square.

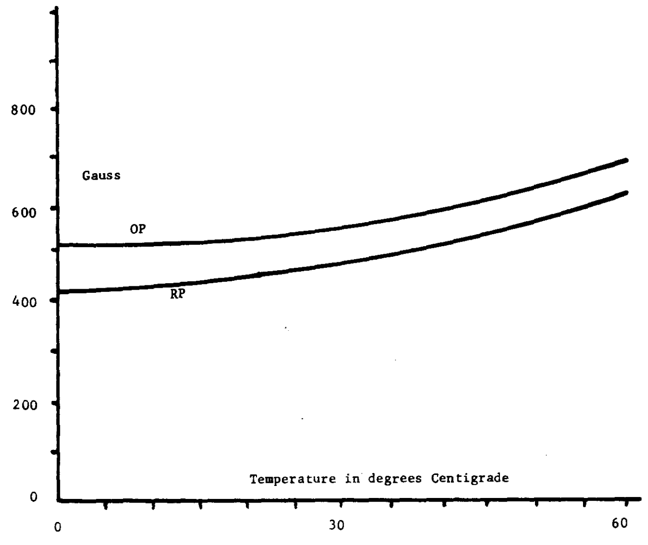


Figure 9—Effect of temperature on magnetic operation

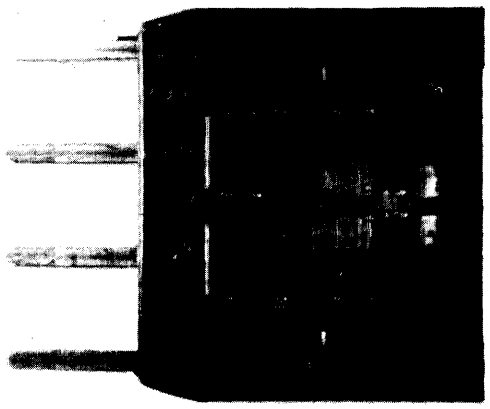


Figure 10—Lead frame showing chip cavity

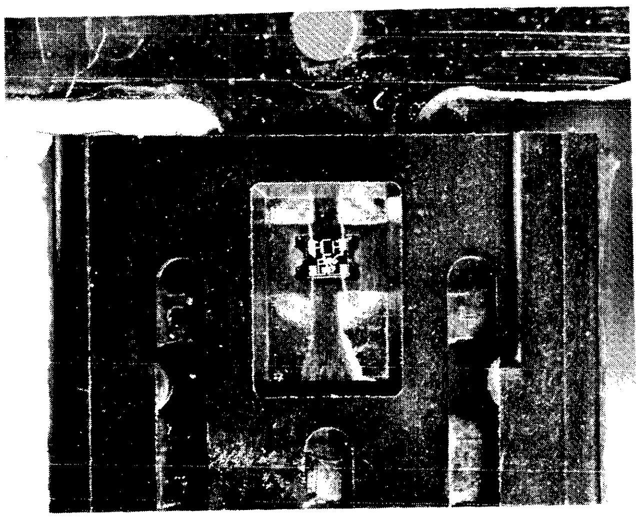


Figure 11—Chip bonded in place

Package thermal considerations

Without a eutectic bond to provide heat transfer between chip and package, it is necessary for heat transfer to occur through the aluminum wires and the silicone potting material. By using .002″ diameter wire the thermal resistance is 355 degrees Centigrade per watt, unpotted. The potting material further reduces this to 266 degrees Centigrade per watt, which is quite comparable to the standard plastic dual in-line package.

Magnet actuation

If a bar magnet is moved along its axis perpendicular to the plane of the Hall element, the normal component of flux will vary with the magnet movement according to the curve shown in Figure 12. Since the curve runs asymptotic to the zero flux axis, a slight change in the release point of the chip would require a large change in the movement of the magnet to reach the release point. This is not desirable. If two magnets are used and are magnetized as shown in Figure 13, the flux versus gap position curve will tend to be sinusoidal. This is desirable if the total travel of the magnet assembly can be limited to the nearly linear portion of the curve. Since the flux required for both operate and release points is positive, the negative portion of the curve would not be used. By inserting the two magnets in a "U" shaped shunt and magnetizing them in place with a specially shaped magetizing fixture, it is possible to produce the curve shown in Figure 14. The result of this is to move the majority of the sinusoid above the zero flux axis. Figure 14 also shows the magnet assembly and the shape of the poles on each of the magnets.

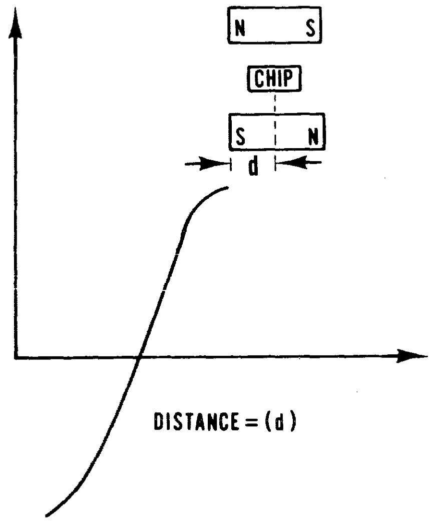


Figure 13—Flux vs position, double pole magnet

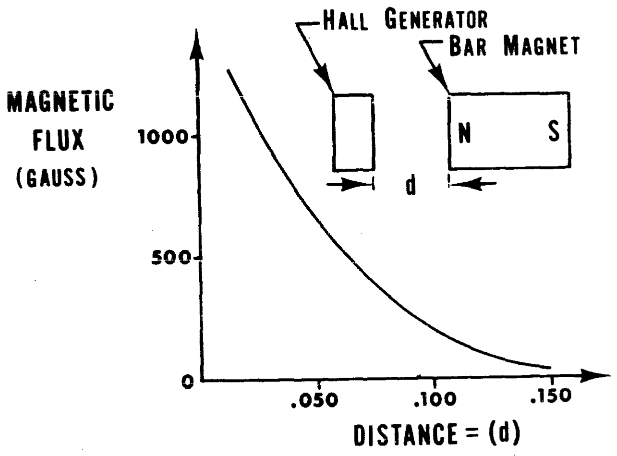


Figure 12—Flux vs position, single pole bar magnet

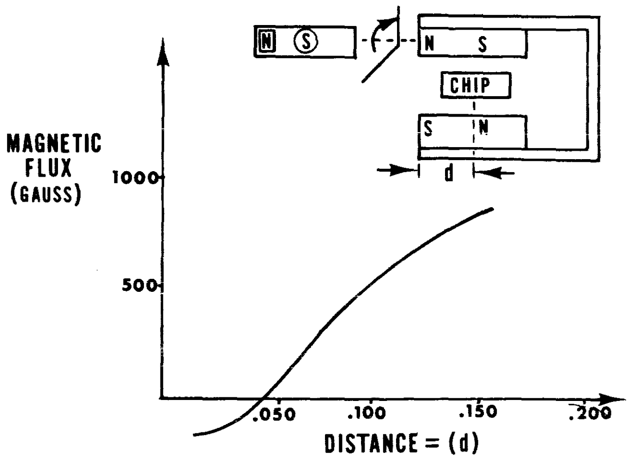


Figure 14—Flux vs position, double pole, modified

The magnets are made of polyvinyl chloride filled with barium ferrite. This combination produces an extremely stable, yet low cost, permanent magnet material. The shunt is soft iron which increases the magnet efficiency and helps to reduce the effect of stray magnetic fields. The chip package is made as thin as possible to reduce the air gap. The magnet assembly with the chip package is shown in Figure 15. Referring to the specification on the chip, and relating these to the flux versus position curve, it is possible to establish the operate and release points of the key, as shown in Figure 16.

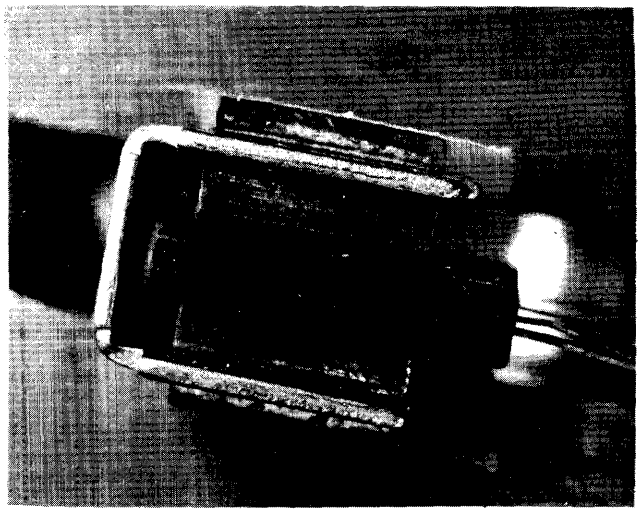


Figure 15—Chip package and magnet assembly

Table I—Chip specifications

Parameter	*Minimum*	*Maximum*	*Units*
Operate Point (OP)	300	750	Gauss
Release Point (RP)	100		Gauss
Differential (OP − RP)	150		Gauss
Supply Voltage	4.75	5.25	Volts
Supply Current (OFF Condition)		15	mA
Output Voltage (ON) @ 5V supply	3.4	3.6	Volts
Output Voltage (OFF) 5000 ohm load		0.25	Volts
Output Current (ON) (each terminal)		10	mA

Reliability test results

A variety of environmental tests have been made on the key chip integrated circuit, packaged as noted herein. In addition to tests on functional performance on conventional chips, chips with special metallization patterns were prepared and packaged, such that junction characteristics and Hall element output could be measured directly. This allows a more sensitive indication of incipient degradation than does functional performance. Table II describes tests on four lots of devices. The results are in keeping with the reliability

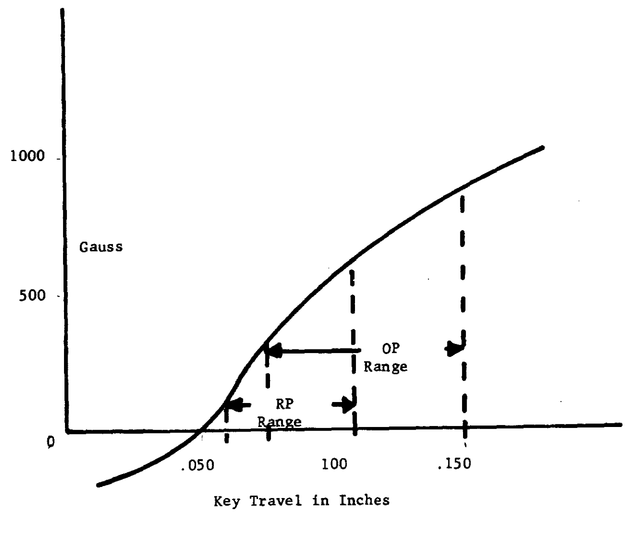


Figure 16—Operate and release ranges of key

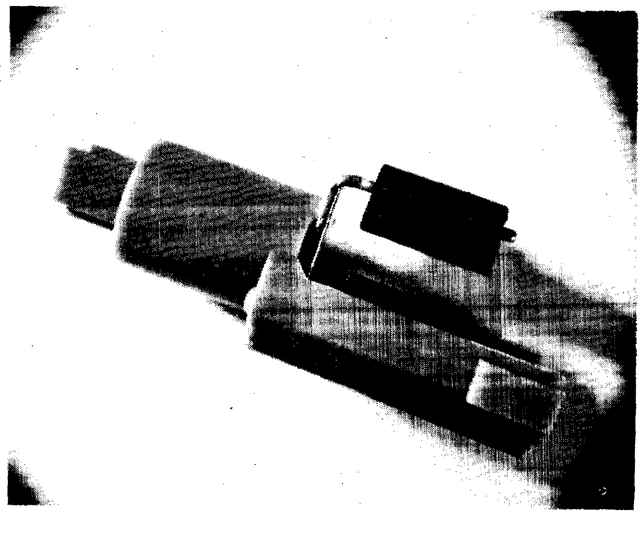


Figure 17—Plunger magnet assembly

Table II—Reliability test results

No. of Devices	*Type of Test*	*Environment*	*Time*	*Results*
30	Functional, magnetic actuation	Normal Office	15 months	No failures
15	Functional, magnetic actuation	75 to 100 deg. F.	4630 hrs.	No failures
30	Hall element (V_q)	70 deg. C. 90% R.H.	1000 hrs.	Maximum variation of 2%
6	Collector junction V_{CBO} @ 10 μA	70 deg. C. 90% R.H.	1000 hrs.	No change

expected of semiconductor devices, when designed, processed and packaged properly. These tests are continuing and others are being initiated.

Mechanical assembly

The magnet assembly is inserted into the key plunger which is shown in Figure 17. The plunger magnet assembly is guided in the key housing by large area guides. We have shaped the top of the plunger and the inside of the two-shot molded button so that the button is press fitted directly into the plunger, avoiding the conventional adaptor pin. In addition to lower cost this provides the advantage of a low keyboard profile.

The chip package is inserted into slots in the housing which hold it in the gap of the magnet assembly. Two small tangs on the bottom of one side of the magnet shunt hold the return spring in place. This spring is designed to provide the two to five ounces of operating force under minimum stress conditions, assuring long life without getting weak.

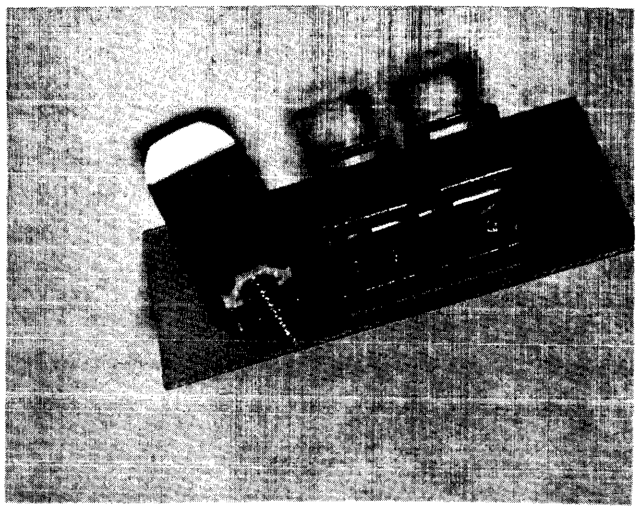


Figure 18—Key assembly

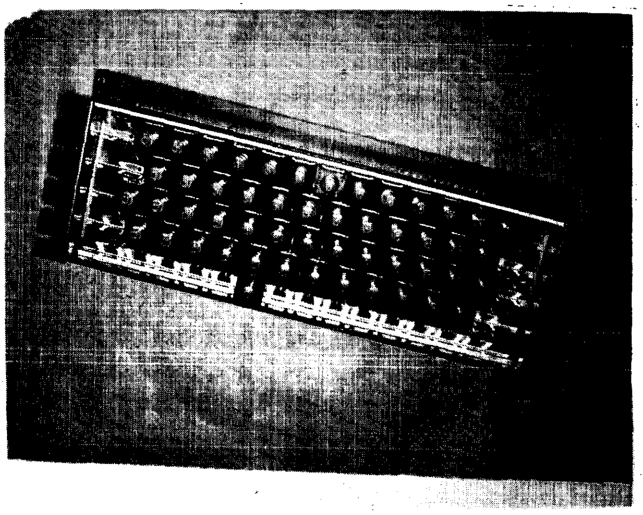


Figure 19—Mounting rail—PC board assembly

The key module, shown in Figure 18, is inserted into a mounting rail. The module snaps into the rail, which has clearance holes for the leads of the chip package to extend through it and be soldered into a printed circuit board. The mounting rails are welded to the end mounting bracket and the entire assembly is riveted to a PC board as shown in Figure 19. The printed circuit board provides the electrical connection between the key modules and a second PC board. The latter contains the electronics for encoding, the strobe signal, and the electrical interlock which prohibits an error code generation when more than one key is depressed.

CONCLUSION

The solid state keyboard uses a new switching concept which capitalizes on the inherent reliability and low cost of integrated circuits. The output of this device is compatible with the integrated circuits used in computers.

The keyboard is deliberately made modular so that it can be adapted to special key formats and codes. It provides an electronic interlock instead of the usual mechanical one, and as a result allows higher speed operation.

While the keyboard is different in many respects, it has maintained those industry standards which have been substantiated by human factor studies such as key stroke and force, key location, and the key layout in the touch typing area.

ACKNOWLEDGMENTS

The development of the solid state keyboard has been possible through the enthusiastic support and dedicated efforts of many people in our respective organizations. We could not hope to fairly cite individual contributions within acceptable space limits here. We also appreciate the consultation provided by other research and engineering groups in our Company.

REFERENCES

1 R L DEININGER
Human factors engineering studies of the design and use of pushbutton telephone sets
BSTJ Vol XXXIX No 4 995–1012 July 1968

2 R L DEININGER
Desirable push-button characteristics
IRE Transactions on Human Factors in Electronics
March 1960

3 H M BOWEN
Rational design for control: Man communicating to machine,
Industrial design Vol XI No 5 51–59 May 1964

4 R D KINCAID
Human factors design recommendations for touch operated keyboards
Report 12091–FR Honeywell Systems and Research Center
January 1969

5 A C BEER
The Hall effect and related phenomena
Solid State Electronics Pergamon Press Vol 9 339–351

6 A C BEER
The Hall effect
International Science and Technology December 1966

7 O N TUFTE E L STELZER
Magnetoresistance in heavily doped N-Type silicon
Phys Rev Vol 139 No 1A A–265–A–271 July 5 1965

8 J G LINVILL J F GIBBONS
Transistors and active circuits
McGraw-Hill Book Co New York 1961

9 R M WARNER JR J N FORDEMWALT (Editors)
Integrated circuits
McGraw-Hill Book Co New York 132–149 1965

10 J T MAUPIN
The control characteristic of current switching logic stages
Honeywell Corporate Research Center Memorandum
HR 63–37 July 1963

Computer generated graphic segments in a raster display

by RICHARD A. METZGER

Rome Air Development Center
Rome, New York

INTRODUCTION

The increased use of computer graphics to enhance the man-machine interface has resulted in many and varied systems and devices to meet a multitude of needs. One type of display that is receiving new emphasis as a computer output device is the "raster format" display (of which standard television is a particular type). Among the reasons for using this type of display are: (1) the relative simplicity of the display device, (2) the ease of remote operation for multiple station users, (3) the low cost per station, (4) capability for mixing output with standard television sources, and (5) good position repeatability for computer generated data.

However, to utilize a raster display (either standard 525 line TV or other line standard) as a computer output device, a conversion from digital to video data must be performed. The device utilized to perform this function is commonly referred to as a Digital-to-Video (D/V) Converter. It accepts digital data from the data processing system and converts it to a video signal compatible with the raster scan display. The converter, being a digital device, often becomes complicated since it is forced to operate within the timing constraints of the Raster Scan Display System (RSDS). A problem also arises in the subjective appearance of the display, since all data must be generated within the line-by-line structure of the raster. In the case of alphanumerics, a fixed-size matrix of dots can be used to generate very acceptable symbology. However, the generation of graphic segments is not as easily accomplished.

Graphic segments and figures vary greatly in terms of size, shape, orientation, and complexity. If the scan lines forming the raster together with the discrete points on each scan line are considered as a Cartesian grid, it can be seen that in general, graphic segments will not fit exactly within the constraints of this grid. In addition, graphic figures require a set of defining equations, each valid for a given domain. Thus one of the simplest means of generating complex figures is by combination of graphic segments (lines, circles, arcs, etc.), into the higher order figures.

The particular problem to be addressed here is the generation of graphic segments (straight and curved lines) within the constraints of a raster format display. Algorithms are developed to allow computer generation of selected graphic segments of arbitrary length (constrained by screen size) at any desired screen location. The raster will be considered as an N x M Cartesian grid, where N is the number of scan lines and M is the number of addressable points on a line. All operations will be performed within the constraints of this address grid. By treating the grid dimensions as variable, the algorithms are immediately usable for any line standard raster. The software approach to development of the algorithms was adopted since this allows usage with any of the standard D-V converters, whereas a hardware implementation is particular as to type. However, there is nothing within the adopted approach which prohibits hardware implementation.

Prior to considering the algorithms for generation of graphic segment, a brief discussion of Digital-to-Video conversion as a display technique will be presented.

Digital-to-video conversion

The generation of data within a raster formatted display is governed by the timing of the raster sweeps. To generate a "dot" at a given point it is necessary to unblank the beam at the instant it traverses the specified address. As stated previously, the Raster Format of the display can be considered as an address coordinate grid where the individual scan lines represent the ordinate values (Y-Address) and the points along the line represent the abscissa values (X-Address).

By considering the relation between the scan rate of the electron beam and the Cartesian address grid, it can be seen that associated with any given picture element there is a corresponding X-Y coordinate address and vice versa. Thus by taking into account the number of lines preceding the addressed line and the number of elements on this line preceding the addressed element, a given time interval after the beginning of each frame can be associated with each picture element. In this way D-V conversion can be considered as a position to time conversion.

Bit per element techniques of D-V conversion

One means of implementing a digital-to-video converter to provide the above type of position to time conversion is the "Bit Per Element" converter. The general functions of such a converter are outlined in Figure 1. Input data is accepted from the data source by the interface control in a word serial, bit parallel form. This data consists of an X and Y address(es), character code and control bits. Data format and parity are checked and if proper, the data is transferred to the buffer memory and process control. The buffer memory is a small high speed digital memory (usually core) which temporarily stores the character, vector, and control data in the same form as received by the interface unit. By means of the process control, the address bits are separated and transferred to the sync and comparator section while the character/vector data is held in the buffer memory. If the data supplied from the data source is not sorted (in X-Y address), then the process control has the additional function of sorting the address data in ascending orders of Y and X within each Y.

Since the raster timing must provide overall control of the conversion as well as display process, all digital and display functions must be timed to the raster synchronization pulses or harmonics thereof. The sync and comparator section provides this control interface as well as provide the necessary timing signals internal to the converter. There are two primary ways of providing the control interface. In the first, the raster synchronization pulses are fed from the display device to the sync and comparator section and the internal timing for the converter derived from it. In the second mode, the sync and comparator section of the converter contains a crystal controlled clock which operates at a harmonic of raster timing, with the latter being derived from the crystal.

Under control of the sync and comparator section, the data is transferred from the buffer memory to the character generator where a dot pattern of the alphanumeric symbol, etc. . . . to be displayed is formed. This dot pattern is then transferred to a section of the video memory determined by the display coordinate address. In the case of a graphic segment, a single dot is generated at each of a series of coordinate addresses with the group or sequence of dots forming the graphic figure.

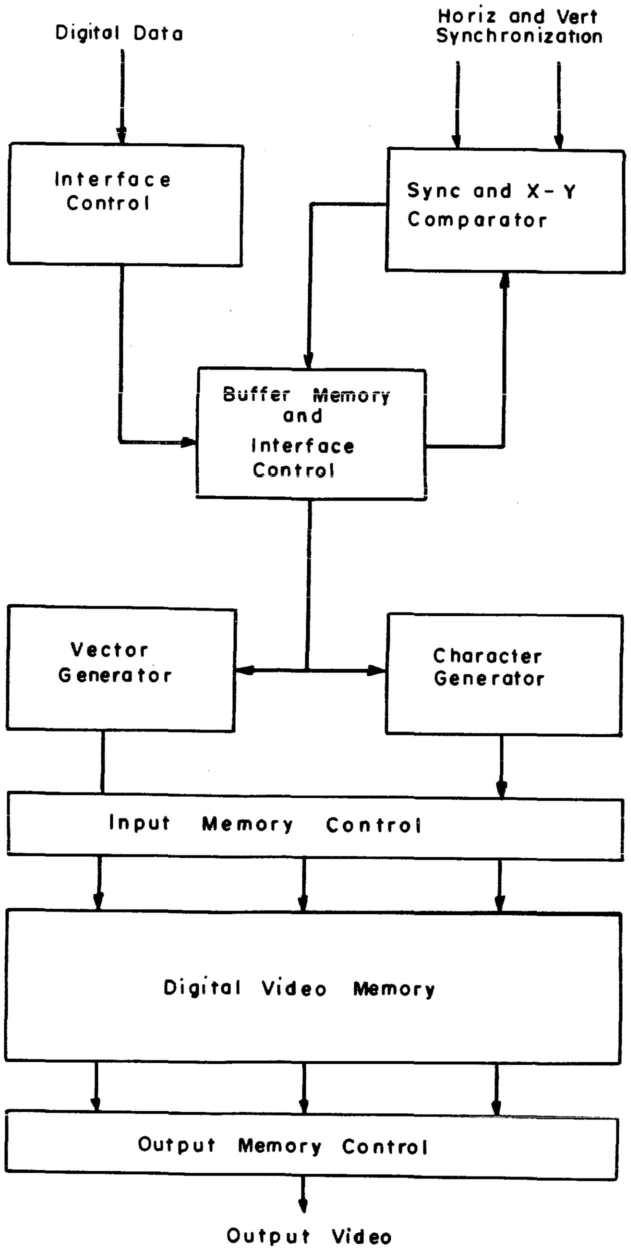


Figure 1—Bit per element digital-to-video converter

The video memory contains one bit of digital storage (1 = unblanked electron beam, 0 = blanked electron beam) for each picture element on the display surface. Thus by loading the dot pattern of the character at a memory address corresponding to the display address, the contents of the video memory bear a one-to-one correspondence to the generated display. To maintain display continuity and eliminate presentation of partial characters caused by loading sections of characters during free memory cycles, loading of the video memory is performed during retrace of the electron beam when it is blanked from the display surface. The memory is read out in synchronization with the beam, i.e., every bit is read out as the electron beam traverses the corresponding point on the display surface. To attain the output speed required, it is necessary to perform multiple word read out, multiplex several tracks or lines, or use very long word lengths. In each case the data is read into a register for parallel to serial shifts. When the serial bit stream is inserted into the synchronization and blanking interval, the video signal results.

Real time techniques of D/V conversion

An alternative method for implementation of D/V converters are the "Real-Time" type represented by Figure 2. The primary difference is the absence of the large (digital) video memory to recirculate the "bit per element" display at the 30 frame/sec refresh rate.

The data is transferred from the data source through the interface unit to the buffer memory in a manner analogous to the previous example. In this case, however, the buffer memory is of sufficient capacity to store a complete frame in computer word form, e.g., to present 1000 characters, each defined by three computer words requires a 3000 word memory. The addressed portion of the stored words are continuously compared to the position of the scanning beam. This is accomplished by use of two counters controlled by the raster synchronization pulses. One counter is advanced by the horizontal synchronization pulse and indicates which scan line (Y-Address) is being written. A second, higher speed, counter advances by M for each horizontal sync pulse, until a number corresponding to the number of picture elements per line is attained. In this way the second counter indicates the picture element (X-Address) being scanned. By continuously sampling both counters, the screen address for any specified X-Y Address can be obtained.

A given interval prior to coincidence (sufficient to account for propagation delay) the synchronization and comparator circuits transfer the character data from the buffer memory to the character/vector generator. A dot pattern of the character/vector is generated at a bit rate sufficient for insertion directly into the raster. Thus the data is transferred directly from the character/vector generator to the display device by means of high speed register without the requirement for a

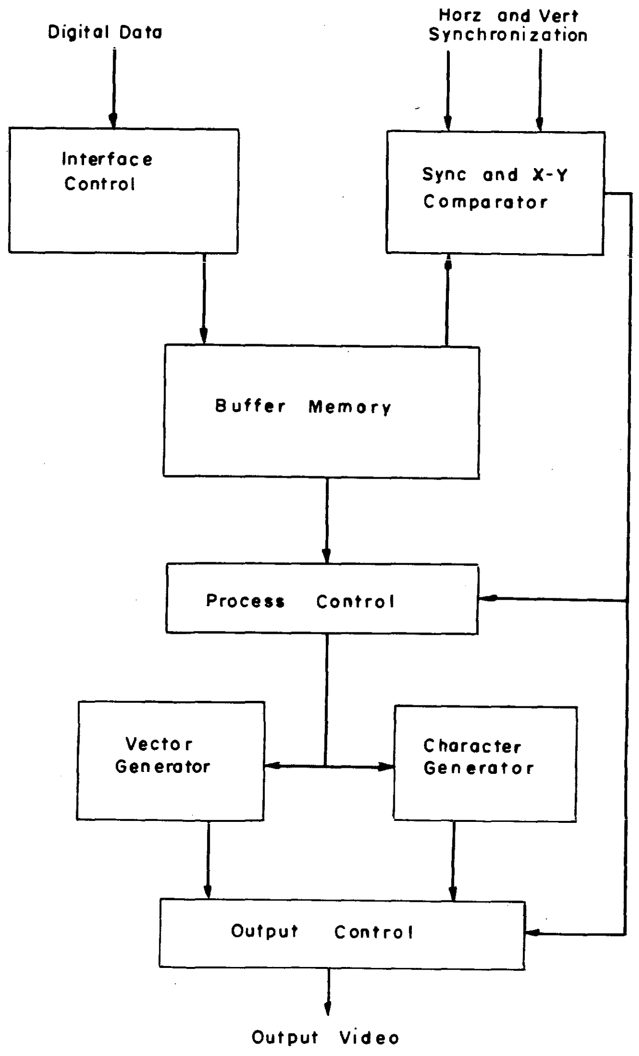
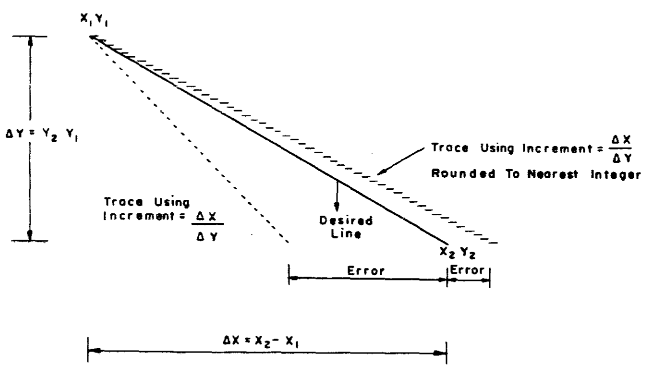


Figure 2—Real time digital-to-video converter

digital video memory. This sequence is performed at a 30 frame per second rate since no digital video memory is available for display refresh. Erasure of the data is accomplished by inhibiting the data output from the character/vector generator and entering new data into the digital memory. In the use of the bit per element converters, a specific erase function (which amounts to entering the complemented character dot pattern) must be provided to remove the displayed dot pattern from the digital video memory.

Straight line generation

The complexity involved in generation of graphic segments in a raster format can be seen by considering the straight line vector as shown in Figure 3. It is desired to generate a line connecting points (X_1, Y_1) and (X_2, Y_2) where the Y values represent raster lines and



X₁Y₁
ΔY = Y₂ Y₁
Trace Using Increment = ΔX/ΔY
Desired Line
Trace Using Increment = ΔX/ΔY Rounded To Nearest Integer
X₂ Y₂
Error
Error
ΔX = X₂ - X₁

Figure 3—Straight line segment in a raster grid

the X values represent picture elements within a raster line. If the lines were horizontal, vertical, or 45 degrees, the generation would be trivial. Without loss of generality, the analysis which follows is for a line directed down to the right at an angle, $\alpha < 45°$ as shown in Figure 3. (A similar type of analysis could be performed for any other octant.)

To form a trace at an angle $\alpha < 45°$, the address of the generated dot must be increased by a given number of units (called the DELTA value), in the X direction, prior to a one-unit increase in the Y address. For a line at an angle $\alpha > 45°$, the address is incremented DELTA units in the Y direction prior to a one-unit increase in the X address. The key to generation of the proper line thus lies in the choice of the "DELTA" value. If DELTA is set equal to the slope,

$$Q = \|\lfloor (X_2 - X_1) \div (Y_2 - Y_1) \rfloor\| \quad (1)$$

then the error is equal to the remainder (R) of the integer division. If the DELTA is set equal to Q rounded to the nearest integer, then the error equals R if $R < \Delta Y/2$ and equals $(\Delta Y - R)$ if $R \geq \Delta Y/2$. In general, if the slope Q is used to determine the DELTA value, then to obtain a zero end point error requires R increments of DELTA equal to $(Q + 1)$ and $(\Delta Y - R)$ increments DELTA equals to Q.
This yields

$$\Delta X = (Q + 1) R + (\Delta Y - R) Q = Q\Delta Y + R \quad (2)$$

which is the condition for zero error. However, it is a formidable task to obtain a line with acceptable linearity and no end point error when computations are based only end point data, since the technique for intermixing the two different length DELTA segments resulting in a zero error is dependent on the particular end points in question. Thus the approach adopted is based upon a "best fit" technique minimizing the error with respect to the desired line, introduced with an incrementation in either the X or Y direction or both.

The main functions to be performed by any straight line algorithms are:

1. determination of direction, right or left, which controls whether incrementation or decrementation, respectively of the X address coordinate is required.
2. determination of whether the angle of inclination is greater than, less than, or equal to 45°.
3. determination of the DELTA segment length.
4. generation of a "dot" at the sequence of points between (X_1, Y_1) and (X_2, Y_2) within the constraints previously listed and in accordance with the above data.

The DELTA value is based upon a determination of whether incrementing the current address (X_s, Y_s), in the X direction, Y direction or both will result in the smallest deviation from the desired line. By summing the number of unit incrementations in a given direction, X or Y (which corresponds to the number of iterations prior to sign change), the DELTA length can be determined.

Let the coordinate grid of Figure 4a represent the raster coordinate grid in the vicinity of the start point X_s Y_s. The actual trace represents the sequence of points which most closely approximates the desired line connecting X_s Y_s and X_f Y_f (not shown). The inclination with respect to the 45° line (determined from ΔX and ΔY) imposes a limitation upon the degrees of freedom of movement. Referring to the case depicted in Figure 4b, it can be seen that the possible new addresses are point 1 $(X_s + 1, Y_s)$ or point 2

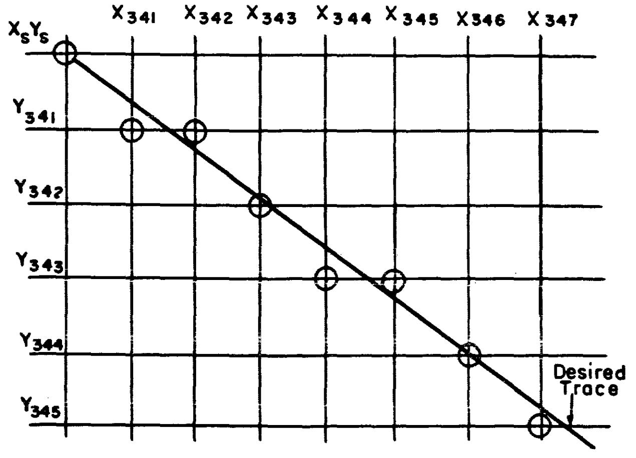


Figure 4A—Coordinate grid as utilized in delta subroutine

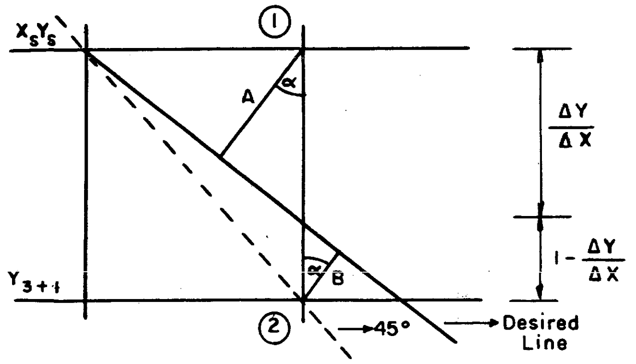


Figure 4B—First move determination in grid structure

$(X_s + 1,\ Y_s + 1)$. If the error associated with each move is represented by A or B respectively, the geometric relationships allow the ratio of the error values to be determined as

$$\frac{B}{A} = \frac{\Delta X - \Delta Y}{\Delta X} \tag{3}$$

If the condition of equal error is chosen as the discriminant point, a value R_1 can be defined such that

$$R_1 = 2\Delta Y - \Delta X \tag{4}$$

where the sign of R_1 indicates the minimum error move to point 1 or point 2. (In Figure 4b the minimum error is to point 2.)

Referring to Figure 4c to consider the second move determination involving error distances C and D, the same analysis yields an R_I discriminant value

$$R_I = R_1 + 2\Delta Y - 2\Delta X \tag{5}$$

If the initial address modification from X_s Y_s had been to point 1 rather than point 2, the R_I for the subsequent move, Figure 4d discriminant, would assume the form

$$R_I = R_1 + 2\Delta Y \tag{6}$$

In both cases the sign of R_I would determine the minimum error move.

If this analysis is pursued to the general case including inclinations both greater and less than 45⁰, the following results are obtained:

a. After the first move from point X_s, Y_s, all subsequent moves reduce to one of the two above cases.
b. The *form* of the error equation is dependent upon

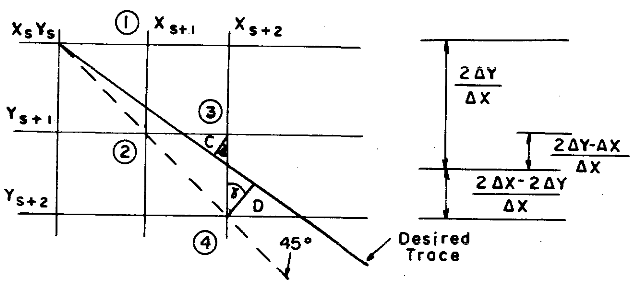

X_sY_s
①
X_{s+1}
X_{s+2}
Y_{s+1}
Y_{s+2}
③
②
C
D
④
45°
Desired Trace
$\frac{2\Delta Y}{\Delta X}$
$\frac{2\Delta Y-\Delta X}{\Delta X}$
$\frac{2\Delta X-2\Delta Y}{\Delta X}$

Figure 4C—Second move determination in grid structure

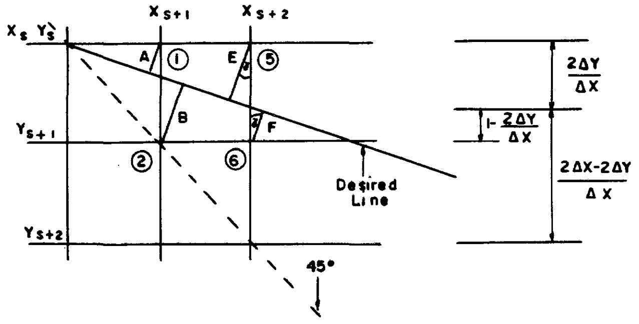

$X_s\ Y_s$
X_{s+1}
X_{s+2}
A
①
E
⑤
B
F
Y_{s+1}
②
⑥
Desired Line
Y_{s+2}
45°
$\frac{2\Delta Y}{\Delta X}$
$1-\frac{2\Delta Y}{\Delta X}$
$\frac{2\Delta X-2\Delta Y}{\Delta X}$

Figure 4D—Alternate second move determination

the previous point with the *value* dependent on both the previous point and the slope.
c. All moves to new addresses can be made, based upon the sign of the error term.

The form of the general equations is based upon inclination:

a. If the angle of inclination is < 45°, then

$$R_1 = 2\Delta Y - \Delta X \tag{7}$$

and

$$R_{I+1} = \begin{cases} R_I + 2\Delta Y - 2\Delta X & \text{if } R_I \geq 0 \quad (8) \\ R_I + 2\Delta Y & \text{if } R_I < 0 \quad (9) \end{cases}$$

b. If the angle of inclination is >45° the roles of X and Y are interposed and we obtain

$$R_1 = 2\Delta X - \Delta Y \tag{10}$$

and

$$R_{I+1} = \begin{cases} R_I + 2\Delta X - 2\Delta Y & \text{if } R_I \geq 0 \quad (11) \\ R_I + 2\Delta X & \text{if } R_I < 0 \quad (12) \end{cases}$$

The resulting equations (Equations 7, 8, 9, 10, 11, and 12) can be employed to determine the length of the DELTA segments by noting the number of iterations performed on $R_I + _1$ between sign changes. Proper initialization based upon inclination with respect to the 45° line allow use of the following equations which are independent of line inclination:

$$R_1 = 2T[2] - T[1] \qquad (13)$$

$$R_{I+1} = \begin{cases} R_I + 2T[2] - 2T[1] & \text{if } R_I \geq 0 \quad (14) \\ R_I + 2T[2] & \text{if } R_I < 0 \quad (15) \end{cases}$$

Each iteration of $R_I \geq 0$ increases the current address by one in both the X and Y directions. Each iteration of $R_I < 0$ increases the X Address by one, while holding the Y Address constant. Summation of the iterations $R_I < 0$ prior to each $R_I \geq 0$ yield the DELTA value.

This algorithm was verified by means of a computer program to generate straight lines connecting any two points within a 525-line raster grid.

The constraints placed on the straight line generator routine were the following:

1. All vectors will be generated in the direction of increasing Y address with the origin of the coordinate system at the upper left-hand corner of the display. (Addresses need not be sorted for input.)
2. All vectors will be processed (although not specified) as one of the following:
   a. Down to the left at an angle <45°
   b. Down to the left at an angle >45°
   c. Down to the right at an angle <45°
   d. Down to the right at an angle >45°
   e. Horizontal line
   f. Vertical line
   g. Down to the right/left at an angle = 45°

Typical results are depicted in Figure 5. The routine required approximately 370 instructions in a machine with a 64 instruction repertoire. The maximum error from any computed point to the desired line is less than 0.5 units.

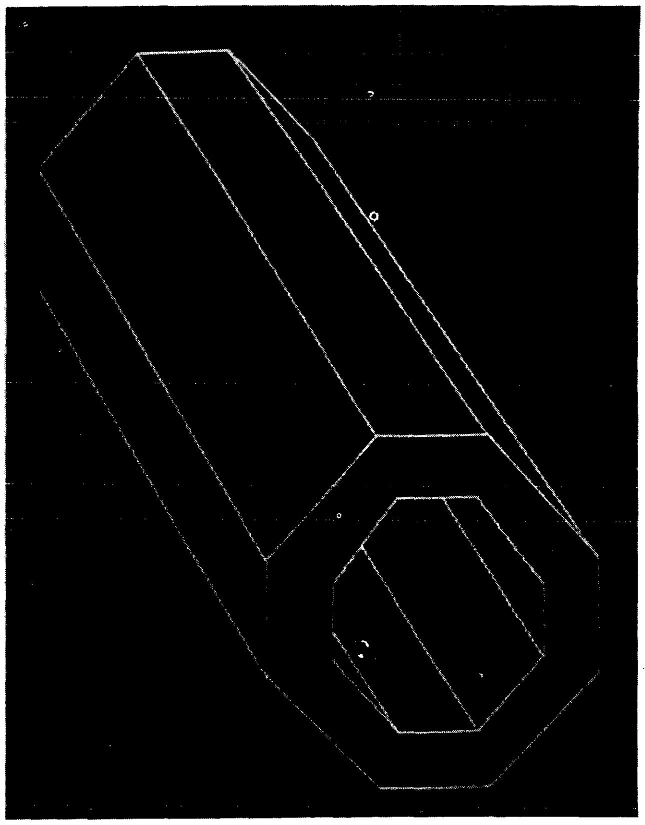


Figure 5—Line segments generated by computer program

Circular line generation

In a manner similar to generation of a straight line, the generation of a circular trace consists of determining the X-Y Address points which minimize the error distance from the desired trace to the selected point. By examining a circular trace overlaid on a Cartesian grid, many of the properties of the circular symmetry are observable. A circle of radius 7 is depicted in Figure 6, together with the appropriate X-Y Address sequence to generate the trace within the constraints of the grid structure, where the Y-Address corresponds to scan lines and X-Address to picture elements along that line. The display trace would be generated by a "dot" (the area of which is equal to a unit square) at each arrowhead.

It can be seen from Figure 7a (and is equally true for any radius) that the address sequence from 1 to A is exactly equivalent to those from 1 to D, 2 to A, 2 to B, 3 to B, 3 to C, 4 to C, and 4 to D. Thus by determining the proper address sequence for the first octant, the address sequence for all other octants (referenced to the center of the circle) have been obtained. It should also be noted that no point is more than one unit removed in each direction from the previous or subsequent point. Two possible address modifications can be determined for each octant, the first of which modifies only one address (X or Y) by one, while the second modifies both X and Y addresses by one. The octant in question determines whether the address modification is an incrementation or a decrementation. The possible address sequence for each octant are depicted in Figure 7b. Using these address sequences, it is apparent that each move encompasses a point exterior to the circle and a point interior to the circle. Table I represents the various possible moves for each octant, where (X_n, Y_n) represents the point from which the move is being made and the columns are the resultant addresses after the move is made.

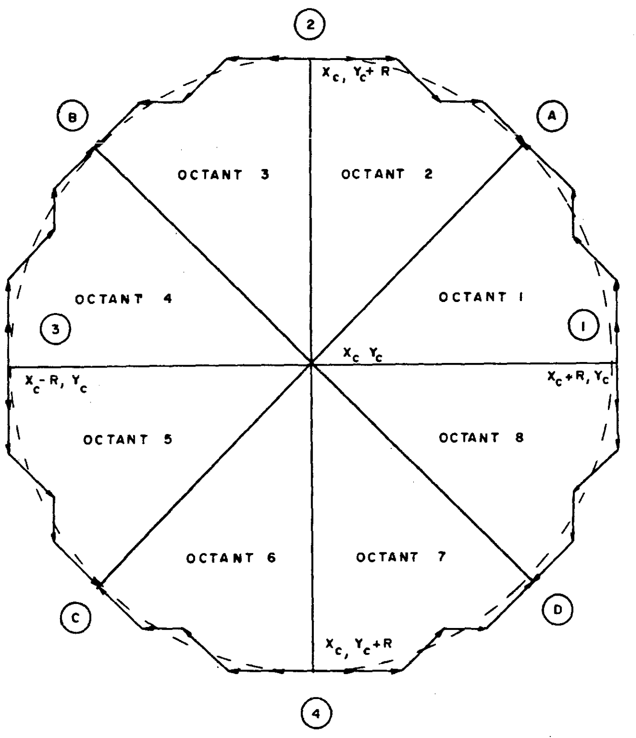


Figure 6—X-Y address sequence for generation of a circular trace

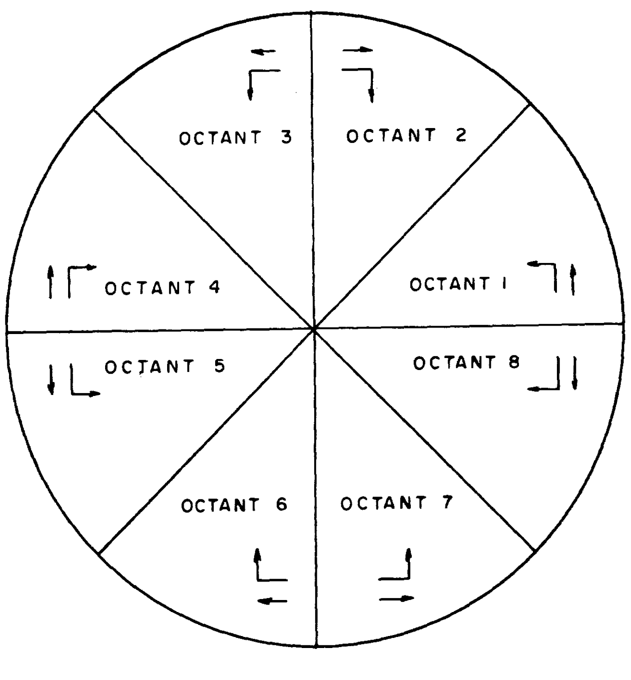


Figure 7A—X-Y address sequence for each octant

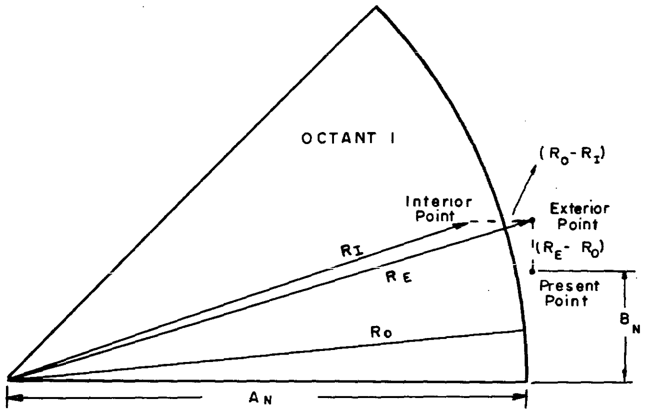


Figure 7B—Computational model for first octant

Table X-Y—Address modification pattern

I.	*Exterior*		*Interior*	
Octant #	X-Address	Y-Address	X-Address	Y-Address
1	X_n	$Y_n - 1$	$X_n - 1$	$Y_n - 1$
2	$X_n + 1$	Y_n	$X_n + 1$	$Y_n + 1$
3	$X_n - 1$	Y_n	$X_n - 1$	$Y_n + 1$
4	X_n	$Y_n - 1$	$X_n + 1$	$Y_n - 1$
5	X_n	$Y_n + 1$	$X_n + 1$	$Y_n + 1$
6	$X_n - 1$	Y_n	$X_n - 1$	$Y_n - 1$
7	$X_n + 1$	Y_n	$X_n + 1$	$Y_n - 1$
8	X_n	$Y_n + 1$	$X_n - 1$	$Y_n + 1$

The problem then is to determine in each case, whether a move to an exterior or an interior point represents the minimum error. This can be accomplished by comparing the differences between the actual radius (R_o) and the external and internal radii (R_e and R_i) respectively, as shown in Figure 7b. If it is determined that

$$(R_o - R_i) > (R_e - R_o) \qquad (16)$$

then the minimum error move is to the next exterior point. Likewise if

$$(R_o - R_i) < (R_e - R_o) \qquad (17)$$

then the interior point represents the minimum error. From Figure 7b it can be seen that

$$R_n \overset{\Delta}{=} \sqrt{A_n^2 + B_n^2} \tag{18}$$

$$R_e = \sqrt{A_n^2 + (B_n + 1)^2} \tag{19}$$

$$R_I = \sqrt{(A_n - 1)^2 + (B_n + 1)^2} \tag{20}$$

Substituting Equations (19) and (20) into Equation (16) yields

$$2R_o > \sqrt{A_n^2 + (B_n + 1)^2} + \sqrt{(A_n - 1)^2 + (B_n + 1)^2} \tag{21}$$

By means of the Triangular Inequality this yields

$$4R_o^2 > (A_n - 1)^2 + (2B_n + 2)^2 \tag{22}$$

In a similar manner, Equations (17), (19), and (20) yield

$$4R_o^2 < (2A_n - 1)^2 + (2B_n + 2)^2 \tag{23}$$

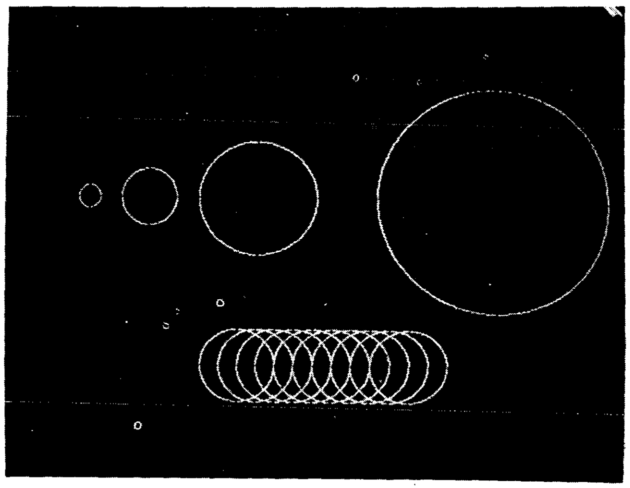
Thus if we define a quantity Q such that

$$Q = (2A_n - 1)^2 + (2B_n + 2)^2 - 4R_o^2 \tag{24}$$

then the sign of Q determines whether the minimum error move is exterior or interior, with $Q < 0$ denoting an exterior point and $Q \geq 0$ denoting an interior point. Due to the symmetry of the trace, it is only necessary to determine the move pattern for the first octant and use this result to perform the address modification for all octants.

This algorithm was implemented and found to yield circular traces (Figure 8) in which no computed element is in error by a distance of more than 0.5 address units in either direction, i.e., interior or exterior. The generation of circular arcs is accomplished by the same algorithm by merely specifying data to define the end points.

It should be noted that for circles with sufficiently small radii (<4 units) the appearance of uniform curvature diminishes. This is due not to a failure of the routine but to the fact that for those radii, insufficient points within the grid structure exist to properly define the curvature. It can be shown that the algorithm itself is valid for all radii greater than 1.25, with this lower limit being dictated by the lowest value of R for which the use of the triangular with Equation 21 remain valid.



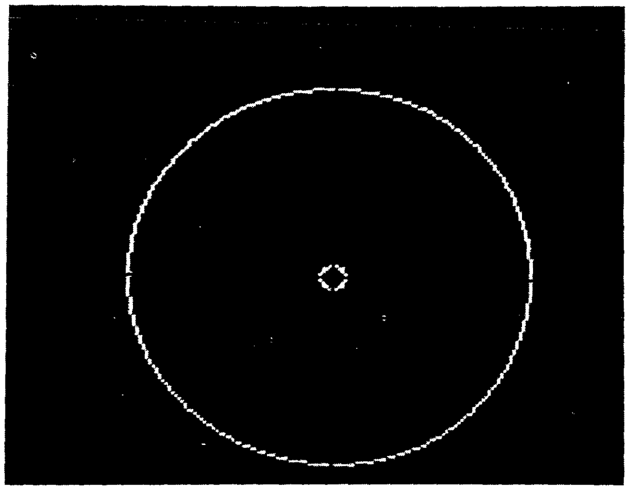


Figure 8—Circular traces generated by computer program

Parabolic line generation

The generation of parabolic trace within a raster format under computer control is accomplished in much the same way as one would generate the trace on linear graph paper. A set of X-Y axes are assumed and for selected values along the Y axis, corresponding values of X are computed according to the defining Equation.

$$X^2 = 4PY \tag{25}$$

Similarly for selected values of X, one can compute values of Y according to the Equation

$$Y^2 = 4PX \tag{26}$$

The difference in method of generation arises in that the trace on the graph paper is obtained by connecting the points defined by the computation, whereas within the raster the increments for the independent variable are chosen to be one line width apart and thus by merely generating a "dot" at the computed X-address points, a continuous curve results.

A typical X-Y address sequence for generation of a parabola is depicted in Figure 9. The actual trace would be created by generating a "dot" at each arrowhead. Although the address sequence will vary depending upon the focii value (P) several important factors can be deduced from the figure. The move pattern is up with increments to the right and left. The number of increments in a given direction is dependent upon the square root for a value of independent variable in relation to the square root for the previous value of the same variable. For each increment in the Y direction if

1. $0 < [\sqrt{4PY_n} - \sqrt{4PY_{n-1}}] \leq \frac{1}{2}$ then $X_n = X_{n-1}$
2. $1/2 < [\sqrt{4PY_n} - \sqrt{4PY_{n-1}}] \leq 1$ then $X_n = X_{n-1} + 1$
3. $1 < [\sqrt{4PY_n} - \sqrt{4PY_{n-1}}] < M$ then $X_n = X_{n-1} + M$

The third case depicted will occur near the base of the parabola with case (1) prevailing as the parabola approaches its asymptote. Since the parabola approaches an asymptote which tends toward infinity, the trace is continued until a screen address is exceeded in one direction at which point the computation stops. The proper incrementation address sequence is dependent upon the form of the equation, the sign of P, and the relative values of the square root for succeeding values of independent variable. The incrementation conditions for all cases are depicted in Table II where (X_n, Y_n) is the point being computed, while the value along axis of symmetry is incremented by one unit.

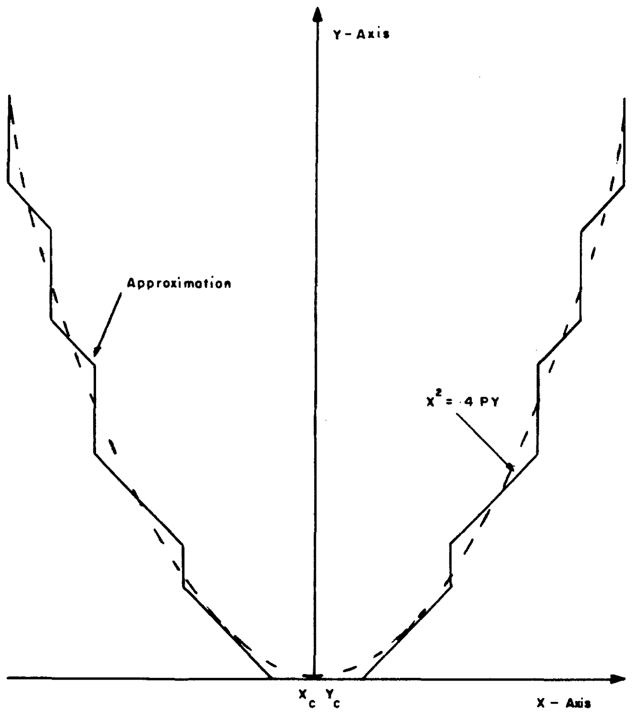


Figure 9—X-Y address sequence for generation of a parabola

The implementation of an algorithm for performing the above arithmetic and manipulative function is not complex. However, the computation time becomes very large for all cases except where the length of the axis of symmetry is kept small. This is necessitated by the iterative methods required for computation of the square root. When implemented on a computer with a six (6) microsecond cycle time, running times in excess of 30 seconds were not uncommon. However, the accuracy obtained placed each computed point within one half display element, since the X (Y) address points are computed for each Y (X) address. The question immediately arises that if the accuracy specification is relaxed and an approximation to the parabola (as it approaches the asymptote) is acceptable, can an appreciable decrease in running time be obtained?

If, for example it is assumed that beyond a given point (X_p, Y_p) (Figure 10) the parabola can be approximated by the straight line (X_p, Y_p), (X_2, Y_2) a maximum error E would result at the point the trace reaches the maximum address or edge of the screen. In addition, to maintain the trace uniformity, the slope of the approximating line is chosen to be the same as the slope of the parabola at the point (X_p, Y_p). The slope of the parabola at X_p, Y_p is

$$\left.\frac{dy}{dx}\right|_{yp} = \frac{X_p}{2P} \tag{27}$$

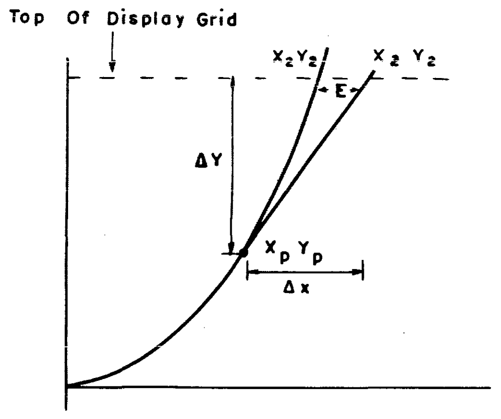
Since this also represents the slope of the approximating line,

then

$$\frac{X_p}{2P} = \frac{\Delta y}{\Delta X} \tag{28}$$

which yields

$$\Delta X = \frac{2P}{X_p}(Y_2 - Y_p) \tag{29}$$



Where Xp and Yp are defined as

$$Xp = \left[(X_2 + E) \pm \sqrt{(X_2 + E)^2 - 4PY_2} \right]$$

$$Yp = \left[(Y_2 + E) \pm \sqrt{(Y_2 + E)^2 - 4PX_2} \right]$$

With the Sign ($\pm$) dependent upon the focii value

Figure 10—Parabolic approximation model

but

$$\Delta X = X_2' - X_p$$

and

$$E = X_2' - X_2$$

Combining the above yields

$$E = X_p - X_2 + \frac{2P}{X_p}(Y_2 - Y_p) \qquad (30)$$

The above equation (30) relates the acceptable error E to the X address of each computed point (X_p, Y_p) and the coordinate of the point at which the parabola reaches the edge of the screen (X_2, Y_2). By allowing the error to be specified by the operator, it allows the choice of the error specification in terms of percent of axis length, percent of opening, absolute units, etc . . . If we use the defining equation for the parabola ($X^2 = 4PY$) together with Equation (30), the value for X_p can be determined as

$$X_p = [(X_2 + E) \pm \sqrt{(X_2 + E)^2 - 4PY_2}] \qquad (31)$$

The value for Y_2 is the STOP address of the axis variable from which X_2 can be computed The approximation can be completed by drawing straight lines between the points (X_p, Y_p) and (X_2 ,Y_2) where $X_2 = X_2 + E$.

A similar analysis can be carried out for a parabola satisfying the Equation

$$Y^2 = 4PX$$

yielding a defining Equation for Y_p,

$$Y_p = [(Y_2 + E) \pm \sqrt{(Y_2 + E) - 4PX_2}]$$

The inclusion of the approximation as a subroutine, to be entered if the allowable error is $E > 0$, results in greatly reduced running times proportional to the error allowed. In addition, the subjective appearance is maintained (Figure 11) by retaining the desired curva-

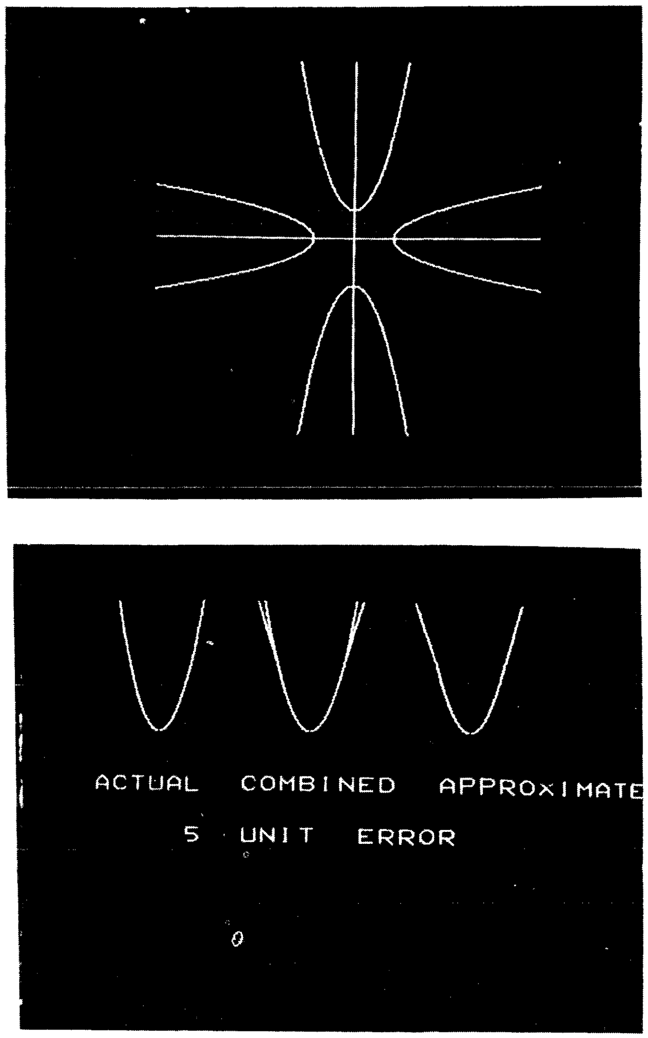


Figure 11—Parabolic traces generated by computer program

Table II—Address modification pattern for parabolic trace

Defining Equation	Axis of Symmetry	P	$X^2 = 4PY$	$Y^2 = 4PX$	Left Side	Right Side	Top	Bottom
$X^2 = 4PY$	Pos. Y Axis	> 0	$> 0, \leq 1/2$	—	$X_n \leftarrow X_{n-1}$	$X_n \leftarrow X_{n-1}$	—	—
"	"	"	$> 1/2, \leq 1$	—	$X_n \leftarrow X_{n-1}{}^{+1}$	$X_n \leftarrow X_{n-1}{}^{+1}$	—	—
"	"	"	$> 1, \leq M$	—	$X_n \leftarrow X_{n-1}{}^{+M}$	$X_n \leftarrow X_{n-1}{}^{+M}$	—	—
$X^2 = 4PY$	Neg. Y Axis	> 0	$> 0, \leq 1/2$	—	$X_n \leftarrow X_{n-1}$	$X_n \leftarrow X_{n-1}$	—	—
"	"	"	$> 1/2, \leq 1$	—	$X_n \leftarrow X_{n-1}{}^{+1}$	$X_n \leftarrow X_{n-1}{}^{+1}$	—	—
"	"	"	$> 1, \leq M$	—	$X_n \leftarrow X_{n-1}{}^{+M}$	${}_n \leftarrow X_{n-1}{}^{+M}$	—	—
$Y^2 = 4PX$	Pos. X Axis	> 0	—	$> 0, \leq 1/2$	—	—	$Y_n \leftarrow Y_{n-1}$	$Y_n \leftarrow Y_{n-1}$
"	"	"	—	$> 1/2, \leq 1$	—	—	$Y_n \leftarrow Y_{n-1}{}^{+1}$	$Y_n \leftarrow Y_{n-1}{}^{+1}$
"	"	"	—	$> 1, \leq M$	—	—	$Y_n \leftarrow Y_{n-1}{}^{+M}$	$Y_n \leftarrow Y_{n-1}{}^{+M}$
$Y^2 = 4PX$	Neg. X Axis	< 0	—	$> 0, \leq 1/2$	—	—	$Y_n \leftarrow Y_{n-1}$	$Y_n \leftarrow Y_{n-1}$
"	"	"	—	$> 1/2, \leq 1$	—	—	$Y_n \leftarrow Y_{n-1}{}^{+1}$	$Y_n \leftarrow Y_{n-1}{}^{+1}$
"	"	"	—	$> 1, \leq M$	—	—	$Y_n \leftarrow Y_{n-1}{}^{+M}$	$Y_n \leftarrow Y_{n-1}{}^{+M}$

ture at the vertex and use of the straight line approximation as the parabola approaches the asymptotic value.

Many other graphic figures could be generated by using the "error minimization" technique above; however, their utility would depend upon the system in question. The figures developed thus far (straight line, circle, circular arc, parabola, and parabolic arc) form the basis for a rudimentary system. The more important aspect from a user point of view would be the combination of the various segment generators for complex figures.

Raster graphic system considerations

To effectively utilize the algorithms thus far defined, they must be incorporated as part of an overall graphic system or subsystem which includes the Graphic Segment Generators as well as a Graphic Operating System.

In configuring the combined Segment Generator, it is assumed that the Operating System would provide the following data in addition to the segment type designator:

a. Straight Line—Start Point (X_1, Y_1), Stop Point (X_2, Y_2)

b. Circle —Center Point (X_c, Y_c), Radius (R)

c. Circular Arc —Center Point (X_c, Y_c), Radius, Start Point (Octant, Y Add) Stop Point (Octant, Y Add)

d. Parabola —Vertex (X_v, Y_v), Focus (P), Orientation (H or V), Axis Stop Point

e. Parabolic Arc—Vertex (X_v, Y_v), Focus, Orientation, Side, Axis Start, Axis Stop

The Generator would be entered from a Graphic Operating System and upon completion of its processing function would return control to the operating system. In addition, all data points would be entered through the Graphic Operating System which would control storage and output of the calculated points as well.

The routine is entered at the same point for any of the segments and branches to the desired section (Straight line, Circle, etc. . .), based upon the mode bits of the data words.

The "move discriminant" functions are the same as previously derived and must remain separate entities in the combined program. In each case, a determination is made to increment or decrement the variables X and Y. In general, N incrementation or decrementation will be performed upon one variable for each increment or decrement of the other variable, where N is calculated by the respective discriminant function.

Typical program length on a six-microsecond, 64-Instruction Set computer, by function:

Functional Sections	*Locations*
Discriminant—Straight Line	60
—Circle	50
—Parabola	70
Incrementation/Decrementation	250
Iteration Control	75
Overhead	350

The size of the output block set aside for storage of the calculated points depends upon the system in question. For example, if the converter being driven requires three computer output words to describe a symbol, then 1200 storage locations would describe 400 symbols to the converter. The size of this storage block is one reason for considering hardware methods of graphics generation as an alternative to software.

The running time for generation of any given segment is directly proportional to the dimensions of that segment since the relative size determines the number of iterations required and all computer instructions execute in equal time.

In the construction of complex figures from simple graphic segments many geometric constructions (Tangents, Normals, etc. . . .) appear often enough to warrant inclusion in the Operating system. In general, this amounts to calculating a new set of defining parameters based upon the segment (or segments) already generated and new segment to be generated.

The following have been investigated and found to produce desired results with minimum software (Figure 12).

1. Head-to-Tail straight lines.
2. Parallel lines.
3. Perpendicular line from a Point X_pY_p on a line (L).
4. Perpendicular line from a Point X_pY_p to a line.
5. Tangent and Normal to a circle.
6. Tangent and Normal to a parabola.

It has been found that if the desired segments are defined in terms of critical parameters much of the manipulative software developed for direct writing systems is applicable, with the segments defined by the manipulated points being generated by the special algorithms.

The entire question of number of segments, hardware vs software generation, amount of manipulative capability, etc. . . , in the final analysis must be determined by the system in question.

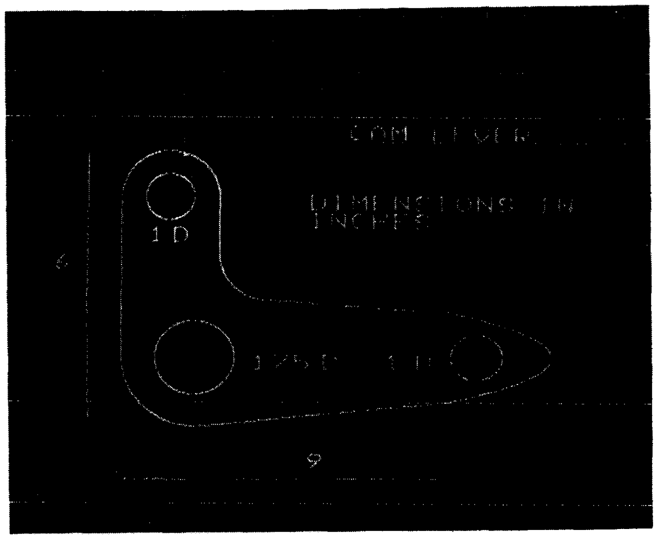


Figure 12—Cam lever generated by graphic segment generator

REFERENCES

1 F G STOCKTON
X-Y move plotting
C A C M Vol 6 No 4 April 1963
2 BECKENBACK & BELLMAN
An introduction to inequalities
Random House New York 1961
3 J H WILKENSON
Rounding errors in algebraic processes
Prentice Hall Series in Automatic Computation
Englewood New Jersey 1963
4 K E IVERSON
A programming language
K Wiley and Co New York 1962
5 F GRUENBERGER
Computer graphics
Thompson Book Co Washington D C 1967
6 L L BRINER
A graphic information processing system
IBM TR-21 197 March 1966
7 I E SUTHERLAND
Sketchpad: A man machine graphical communication system
Technical Report #296 Lincoln Labs MIT Jan 1963
8 A D FALKOFF & IVERSON
The APL/360 terminal system
IBM Research Report RC-1922 Oct 16 1967
9 A D FALKOFF & IVERSON
The APL terminal system: instructions for operation
IBM Research Yorktown Heights New York 1966

Errors in frequency-domain processing of images *

by GRANT B. ANDERSON and THOMAS S. HUANG

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

Practical techniques for the determination of image spectra have been developed and become popular in the past few years. Both optical processing systems and digital computers can be used to perform linear filtering via the frequency domain. Optical processing systems use Fourier-transforming lenses and coherent light. Digital computer software uses the Cooley-Tukey algorithm to advantage, while computer hardware must be augmented by optical scanning devices that interface with images. Processing errors arise in both types of systems, but for different reasons. In this paper we present some results concerning errors in the spatial frequency domain.

Two-dimensional Fourier analysis

To facilitate later discussions, we shall review briefly the key relations in two-dimensional Fourier analysis. The Fourier transform of f(x, y) is defined as

$$F(u, v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi(ux+vy)}\, dxdy \,. \quad (1)$$

The inversion relation is then given by

$$f(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u, v)\, e^{j2\pi(ux+vy)}\, dudv \,. \quad (2)$$

If f(x, y) is nonzero only inside a finite rectangular area $0 \leq x \leq T_x$, $0 \leq y \leq T_y$, then a two-dimensional Fourier series may be used to represent f(x, y) in that area. In particular,

$$f(x, y) = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} \alpha_{m,n}\, e^{j2\pi(mx/T_x+ny/T_y)} \,. \quad (3)$$

$$\alpha_{m,n} = \frac{1}{T_xT_y}\int_0^{T_x}\int_0^{T_y} f(x, y)\, e^{-j2\pi(mx/T_x+ny/T_y)}\, dxdy. \quad (4)$$

When f(x, y) is impulsive and of the form

$$f(x, y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \beta_{m,n}\, \delta(x - m\tau_x, y - n\tau_y), \quad (5)$$

where δ is the Dirac delta function and $T_x = M\tau_x$ and $T_y = N\tau_y$, then a discrete Fourier transform is appropriate. The discrete Fourier transform of the discrete function $\beta_{m,n}$ is given as

$$\lambda_{k,i} = \frac{1}{\sqrt{MN}}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \beta_{m,n}\, e^{-j2\pi(km/M+in/N)}, \quad (6)$$

where k = 0, 1, . . ., M − 1, and i = 0, 1, . . ., N − 1. The inversion relation is

$$\beta_{m,n} = \frac{1}{\sqrt{MN}}\sum_{k=0}^{M-1}\sum_{i=0}^{N-1} \lambda_{k,i}\, e^{j2\pi(mk/M+ni/N)}. \quad (7)$$

Fourier transformation

Optical processing systems[1]

When a film transparency of complex amplitude f(x, y) is illuminated with collimated monochromatic light at the front focal plane of a double-convex lens, the light amplitude at the back focal plane will be

* This work was supported principally by the National Institutes of Health (Grants 5 PO1 GM-14940-02 and 5 PO1 GM-15006-02), and in part by the Joint Services Electronics Program (Contract DA28-043-AMC-02536(E)).

$F(x/\lambda d, y/\lambda d) = F(u, v)$, the Fourier transform of $f(x, y)$. In this relation, x and y are spatial coordinates, λ is the wavelength of light, d is the focal length of the lens, and u and v are spatial frequencies.

Digital processing systems

The Cooley-Tukey algorithm reduces the number of basic operations in the calculation of discrete Fourier transforms from N^2 to $2N \log_2 N$, where N is the number of sample points involved, and a basic operation is defined to be a complex multiplication followed by a complex addition.[2] This time-saving reduction has made the calculation of image spectra practical for digital machines.

A device, which can act as an interface between images on film and the digital computer, is needed as auxiliary equipment. Precision flying-spot scanners, such as the one built by Professor W. F. Schreiber at M.I.T., are ideal for this purpose.[3]

Linear filtering

Optical processing systems

The simplest optical processing system capable of doing linear filtering uses two double-convex lenses and two film transparencies aligned along a path of collimated coherent light. An input film is placed at the front focal plane of the first lens, while the filter-function transparency is placed at the back focal plane of the first lens. The front focal plane of the second lens is coincident with the back focal plane of the first lens. When no errors are present, the light amplitude at the back focal plane of the second lens, except for a possible change of scale, is

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)\, H(u, v)\, e^{j2\pi(ux+vy)}\, du dv, \tag{8}$$

where H(u, v) is the filter function, and F(u, v) is the Fourier transform of the input image.

Complex frequency-domain filters for optical filtering may be made by varying the density of film according to a desired magnitude function and varying the film thickness to regulate phase. Practical difficulties in varying film thickness in this direct method have led to the development of the methods of Vander Lugt,[4] Lohmann,[5] and Lee.[6] These methods permit complex filtering using positive real filters. In the Lohmann and direct methods, noise in the filter transparency can be modeled approximately as independent white noise that is being added to the magnitude and phase of the filter function. For the methods of Vander Lugt and Lee, noise in the filter transparency can be modeled approximately as independent noise that is being added to the real and imaginary parts of the filter function.

Vander Lugt used a reference beam of coherent light in recording his complex filter on film. The function recorded on film, which is non-negative, is

$$\begin{aligned} S(u, v) &= |A\, e^{j2\pi au} + H(u, v)|^2 \\ &= A^2 + |H(u, v)|^2 + AH^*(u, v)\, e^{j2\pi au} \\ &\qquad + AH(u, v)\, e^{-j2\pi au}, \end{aligned} \tag{9}$$

where the asterisk denotes complex conjugation. The impulse response of S(u, v) is

$$\begin{aligned} s(x, y) &= A^2\delta(x, y) + R_h(x, y) + Ah(-x - a, -y) \\ &\qquad + Ah(x - a, y), \end{aligned} \tag{10}$$

where h(x, y) is the impulse response of H(u, v), and $R_h(x, y)$ is the autocorrelation function of h(x, y). Multiplication of F(u, v) by S(u, v) corresponds to the convolution of f(x, y) with s(x, y). If f(x, y) and h(x, y) are of finite spread, then the constant a can be chosen to produce the desired output g(x, y) without interference, but displaced along the x-axis in the output plane.

The filter of Lohmann has only binary transmittance values. Clear slits for light transmission are placed on film to synthesize complex transmission functions. The slit area determines the magnitude of light transmission. Varying the slit position changes the phase of the light transmitted through the slit.

Lee's method for producing complex filters on film is similar to Lohmann's method, except that it provides for a continuous variation in transmittance. Lee's filter uses four non-negative sample points placed along a line to construct a complex sample point. The positions of the four sample points result in transmission phases of 0, $\pi/2$, π, and $3\pi/2$, respectively. By adjusting the transmission amplitudes for the four points, any desired complex transmittance can be obtained.

Digital processing systems

Linear filtering is accomplished on the digital computer by Fourier transformation, followed by multiplication, followed by Fourier inversion. Round-off error occurs in this process. A crude model for the error is independent white noise added during the computation.

Quantitative effects of frequency-domain errors

Additive noise

If independent noise is added to the real and imaginary parts of F(u, v) [Equation (2)], then independent noise of the same power will be present in the reconstructed image f(x, y) upon Fourier inversion. This is true because of Parseval's theorem. In particular, independent white noise transforms to independent white noise of the same power. Independent white noise added to the magnitude of F(u, v) also results in contamination of f(x, y) by independent white noise. The model of independent white noise can be used as a first approximation for grain noise in film, and for round-off error in digital computations.

Multiplicative noise

When Vander Lugt or Lee filters are used for linear filtering, g(x, y) [Equation (8)] is contaminated by a noise equal to

$$n(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)\, N(u, v)\, e^{j2\pi(ux+vy)}\, dudv\ , \tag{11}$$

where N(u, v) is the film grain noise added to H(u, v), and is assumed to be independent. Under the condition

$$\overline{N(u, v)\, N^*(a, \beta)} = \sigma^2 \delta(u - a, v - \beta), \tag{12}$$

where σ^2 is a positive constant, and the bar denotes ensemble average, it may be shown that

$$\overline{n(x, y)\, n^*(r, s)} = \sigma^2 R_f(x - r, y - s)\ , \tag{13}$$

where R_f is the autocorrelation function of the input image f(x, y). Film grain noise causes errors in |H(u, v)|, the magnitude of H(u, v), for the direct method filter, while inaccuracy in the slit area has the same effect with the Lohmann filter. In these cases the filter becomes

$$\hat{H}(u, v) = \{|H(u, v)| + N(u, v)\}\, e^{j\phi(u,v)}\ , \tag{14}$$

where $\phi(u, v)$ is the phase of H(u, v). If N(u, v) is independent and obeys Equation (12), then the noise in the filtered image $\check{g}(x, y) = g(x, y) + n(x, y)$ is given by

$$n(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)\, N(u, v)\, e^{j\phi(u,v)}\, e^{j2\pi(ux+vy)}\, dudv \tag{15}$$

and we again have

$$\overline{n(x, y)\, n^*(r,s)} = \sigma^2 R_f(x - r, y - s)\ . \tag{16}$$

Phase noise

Spurious film thickness variations cause errors in $\phi(u, v)$ for filters made by the direct method. Inaccuracy in the slit positions has the same effect in the Lohmann method.

When phase noise occurs, the filter function will be given by

$$\hat{H}(u, v) = |H(u, v)|\, e^{j[\phi(u,v)+N(u,v)]}\ . \tag{17}$$

The output of the filtering system then is

$$\hat{g}(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)\, H(u, v)\, e^{jN(u,v)}\, e^{j2\pi(ux+vy)}\, dudv\ . \tag{18}$$

The noise output of the system is

$$n(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v)\, H(u, v)\, [e^{jN(u,v)} - 1]\, e^{j2\pi(ux+vy)}\, dudv\ . \tag{19}$$

If the noise is small ($N \ll 1$), independent, and satisfies Equation (12), then it follows that

$$\overline{n(x, y)\, n^*(r, s)} \cong \sigma^2 R_g(x - r, y - s)\ , \tag{20}$$

where R_g is the autocorrelation function of g(x, y), the ideal output image.

Under the condition that N(u, v) is Gaussian, Anderson[7] has shown that

$$\overline{|n(x, y)|^2} = 2R_g(0, 0)\, (1 - e^{-\sigma^2/2})\ , \tag{21}$$

where σ^2 is the phase noise power measured in (rad/sec)^2. Under the assumption that $\phi(u, v)$ is uniformly

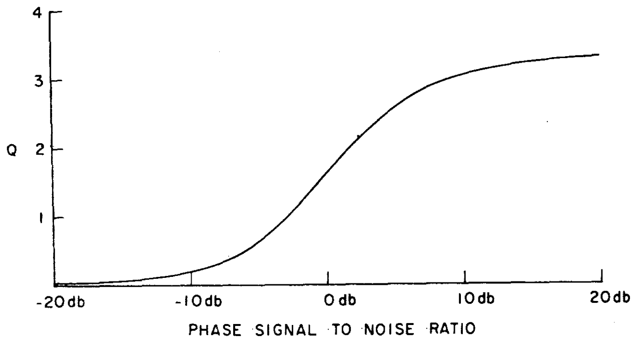


Figure 1—Q as a function of phase signal-to-noise ratio

distributed from $-\pi$ to π, the phase function $\hat{\phi}(u, v) = \phi(u, v) + N(u, v)$ has a signal-to-noise ratio

$$(S/N)_{\phi}^{\times} = \frac{\pi^2}{3\sigma^2}. \tag{22}$$

It follows from Equations (21) and (22) that the ratio of phase signal-to-noise ratio to image signal-to-noise ratio is

$$Q = \frac{2\pi^2}{3\sigma^2}\left(1 - e^{-\sigma^2/2}\right). \tag{23}$$

This ratio is plotted in Figure 1 against phase signal-to-noise ratio.

Quantization noise

Images on film have limited brightness ranges, because of film characteristics. Fourier-transforming functions that represent film brightness variation normally lead to functions with much wider dynamic range. Typically, television quality images have a 30–50 dB disparity between the energy in the lowermost

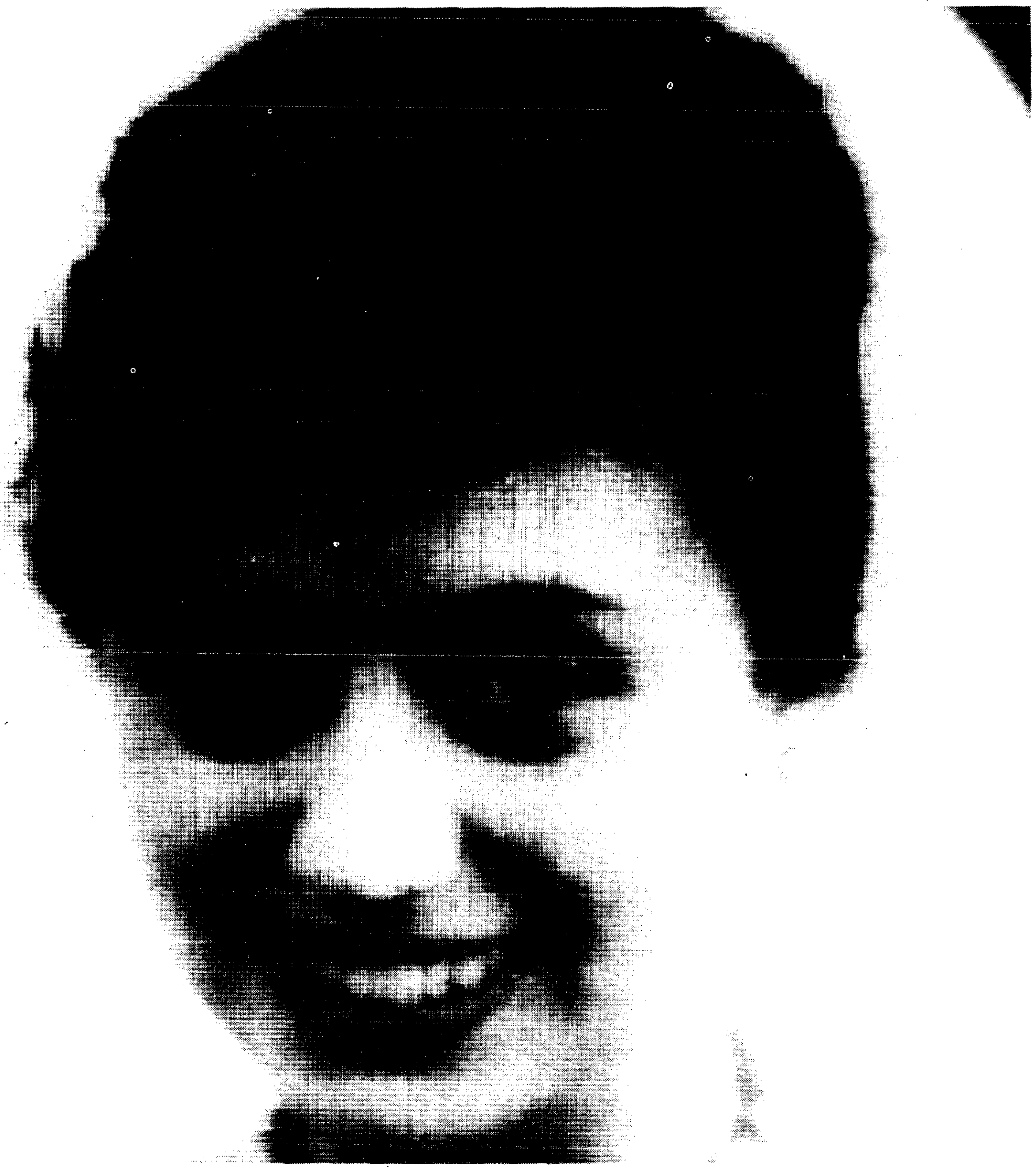


Figure 2—Test image

spatial frequencies and the energy in the highest spatial frequencies of the spectrum. Most of the energy tends to reside in a small area of the total spectrum and is low-frequency. Linear quantization in the frequency domain without a fine quantization grain therefore causes not only a large mean-squared image error, but also results in high percentage errors in the middle and high frequencies. Since it is known that the response of the human visual system is poorest at low and high spatial frequencies and peaks at middle-range frequencies,[8] we can assume that linear quantization in the frequency domain will yield images of poor quality. Improvement in mean-squared error and picture quality can be obtained by using nonlinear quantization with smaller quantization intervals at low amplitude levels and larger intervals at high amplitude levels.

Subjective effects of frequency-domain errors

Experimental system

The human visual system is highly nonlinear.

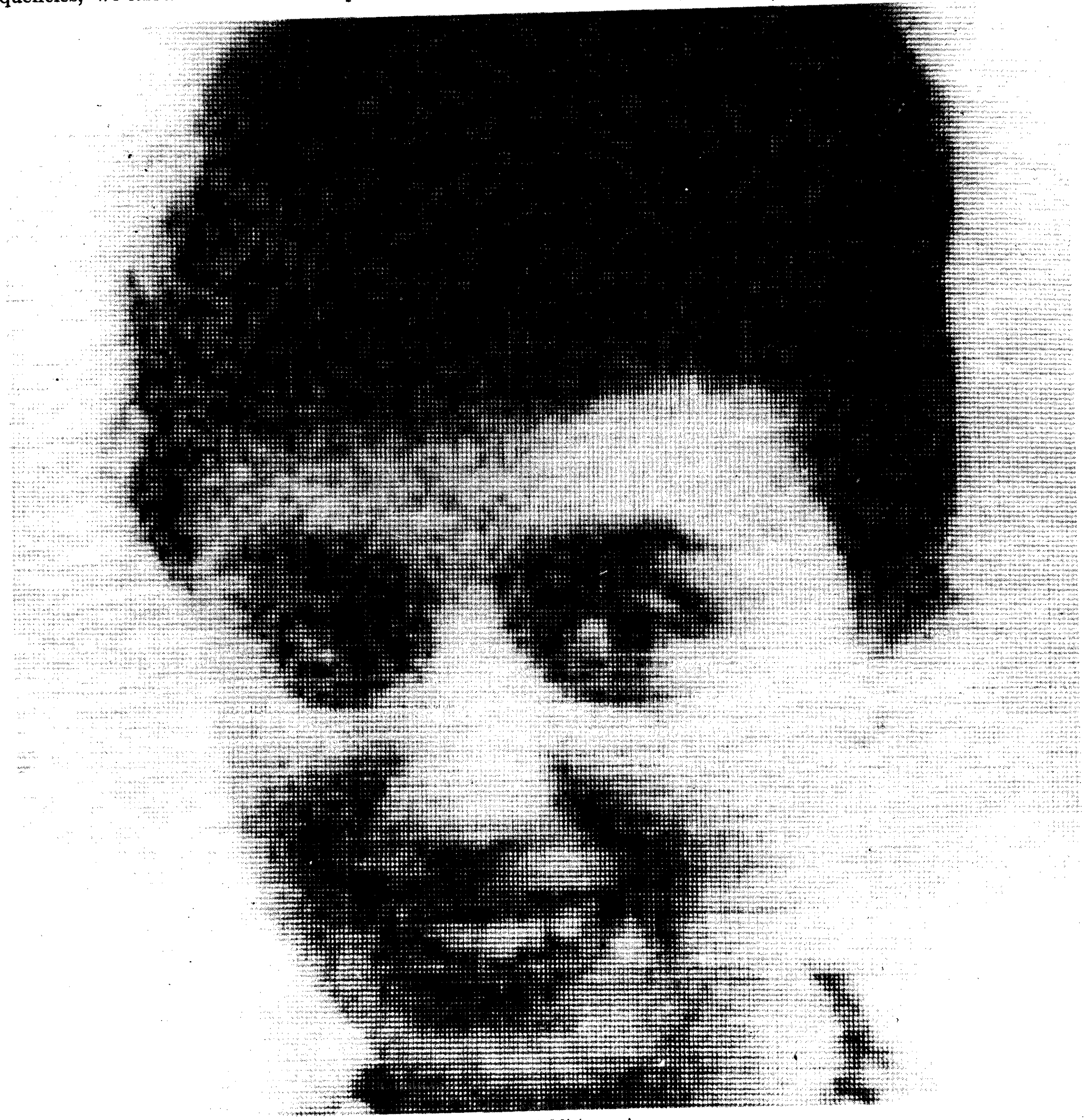


Figure 3—Additive noise

Although, for some purposes, quantities such as mean-squared error and resolution are useful in describing images, it is recognized that good parameters have yet to be discovered to describe image quality. With this in mind, the simulation of frequency-domain error situations was undertaken.

We recorded the test image (Figure 2) on magnetic tape, using the flying spot scanner built by Professor William F. Schreiber of the Research Laboratory of Electronics, M.I.T. Brightness was quantized to 8 bits in the process, and a sample array size of 128 × 128 samples was used. The noise environments were simulated on the IBM 360-65 general-purpose digital computer, and the data recorded on magnetic tape provided an input. Before display the processed outputs were extended to 256 × 256 arrays by using two-dimensional linear interpolation.

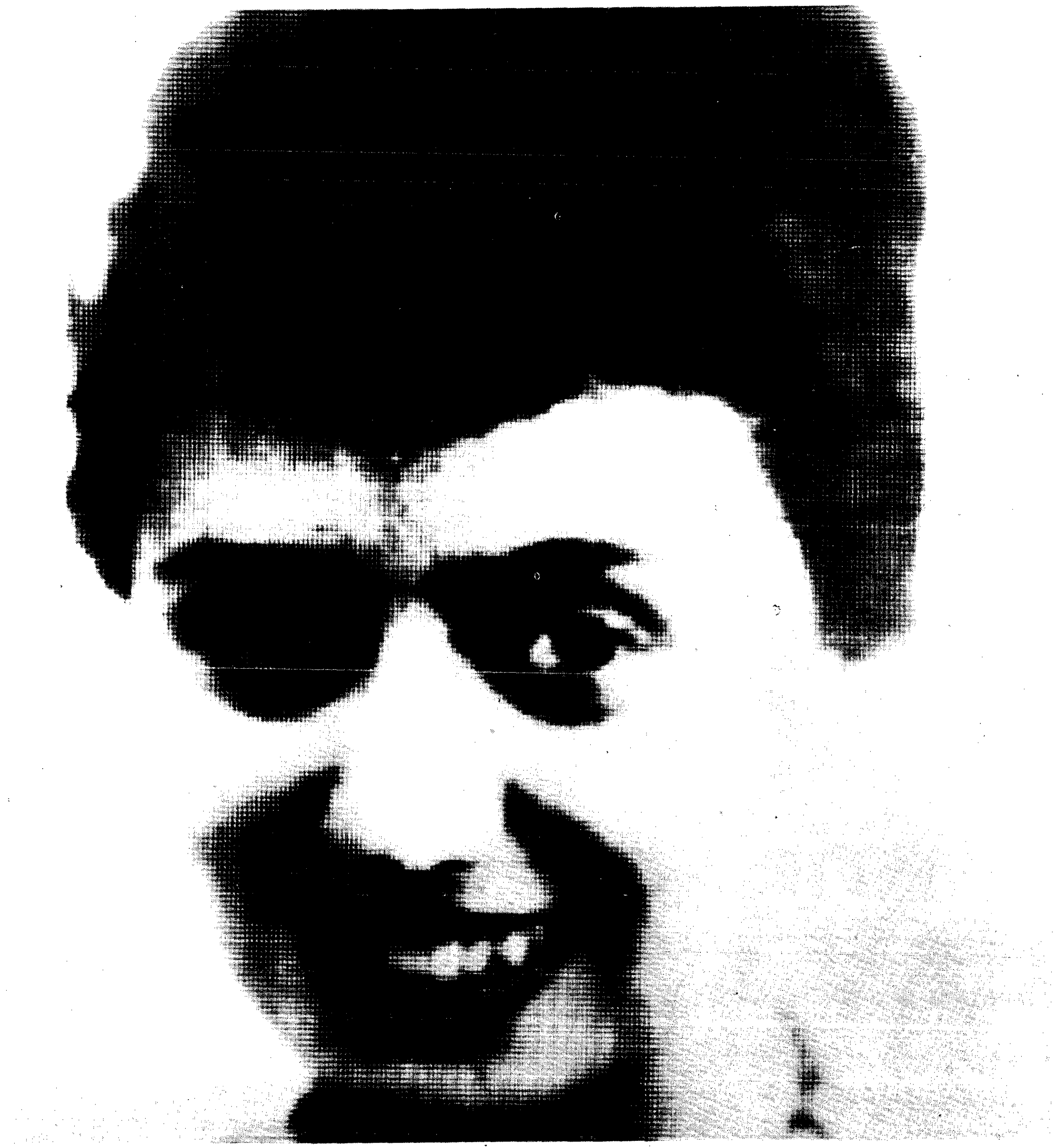


Figure 4—Multiplicative noise

Results

The image in Figure 3 results when independent white noise with a Gaussian probability density is added in the frequency domain. This noise addition, as can easily be shown, is statistically equivalent to adding the same noise directly to the image brightness function. Figure 4 is an image corrupted by multiplicative frequency-domain noise, which is white, independent, and Gaussian. Phase noise has altered the input image to produce the image of Figure 5. The phase noise is also white, independent, and Gaussian. What is interesting about the images of Figures 3–5 is that the signal-to-noise ratio is the same for all noise additions, 15 dB.

The image in Figure 6 is a 5-bit, linearly quantized version of the input image. To compare this image with linear quantization in the frequency domain, Figure 7 is given. The spectrum magnitude and phase of the image of Figure 7 were both linearly quantized to 5 bits.

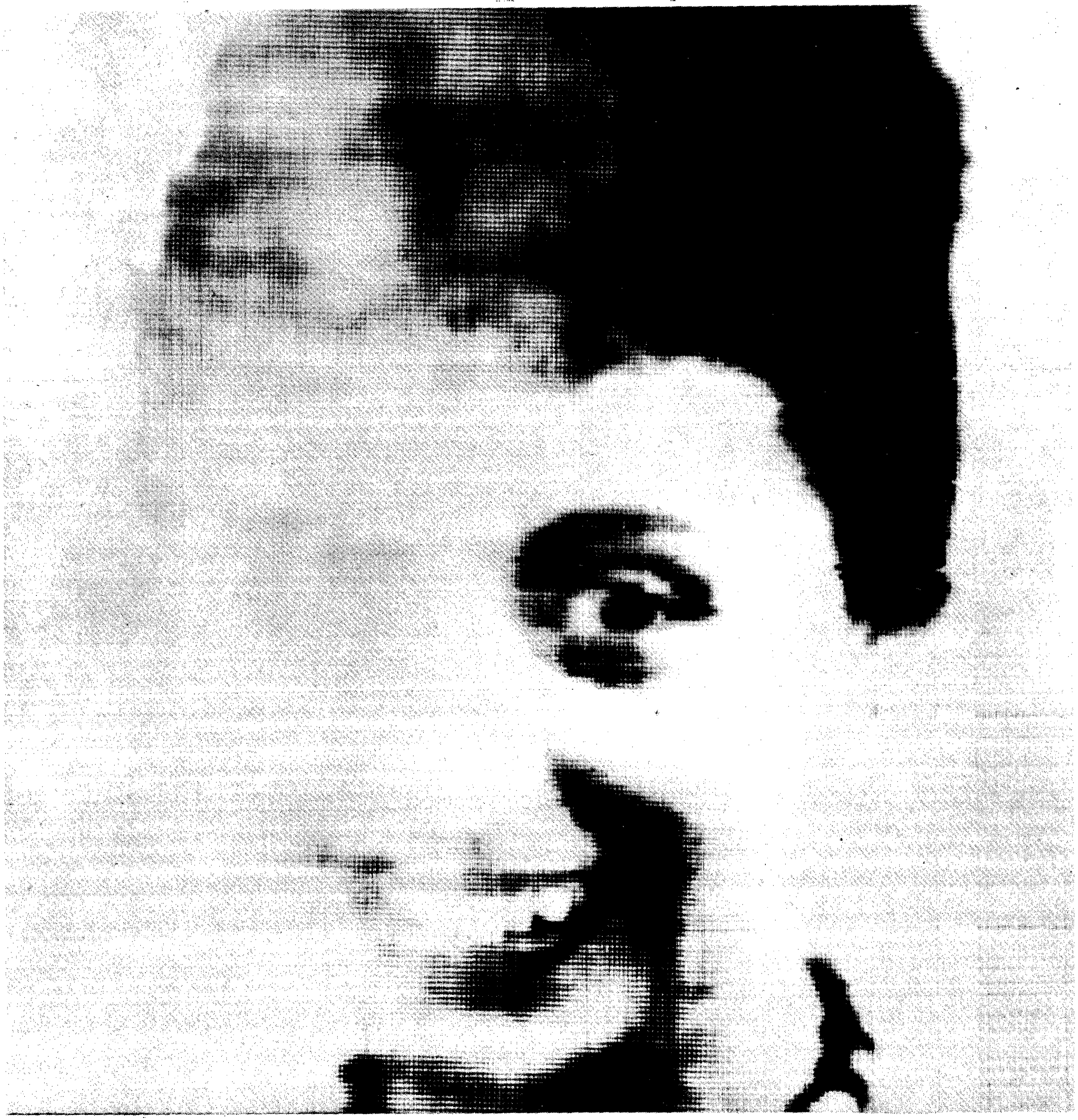


Figure 5—Phase noise

A nonlinear quantization of the spectrum magnitude can be performed to improve the quality of this image. Choosing quantization intervals on a logarithmic scale for magnitude quantization and retaining linear quantization for phase yield the image of Figure 8, which is also a 5-bit image. The signal-to-quantization-noise ratios in Figures 7 and 8 were measured and found to be 9.78 and 13.90 dB, respectively. The images in Figures 9 and 10 are presented to illustrate the effects of magnitude and phase quantization separately. The image in Figure 9 has a spectrum magnitude that is quantized to 3 bits on a logarithmic scale, while phase has been undisturbed. In the case of the image in Figure 10, the phase has been uniformly quantized to 3 bits, while the magnitude has not been changed.

The dynamic range of an image spectrum can be partially characterized by the one-dimensional functions $|F(u, o)|$ and $|F(o, v)|$. These functions are plotted

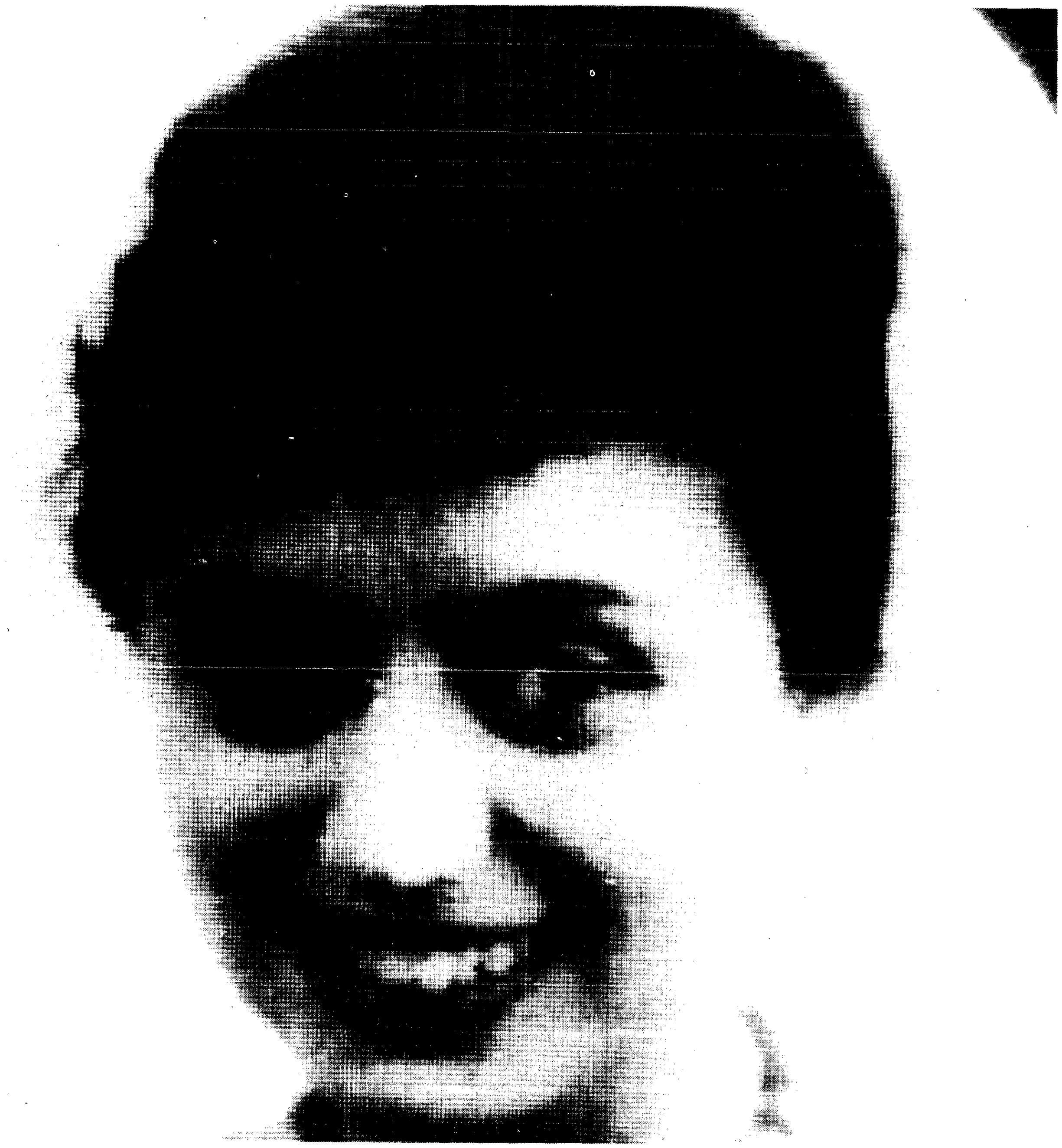


Figure 6—Linear brightness quantization (5-bits)

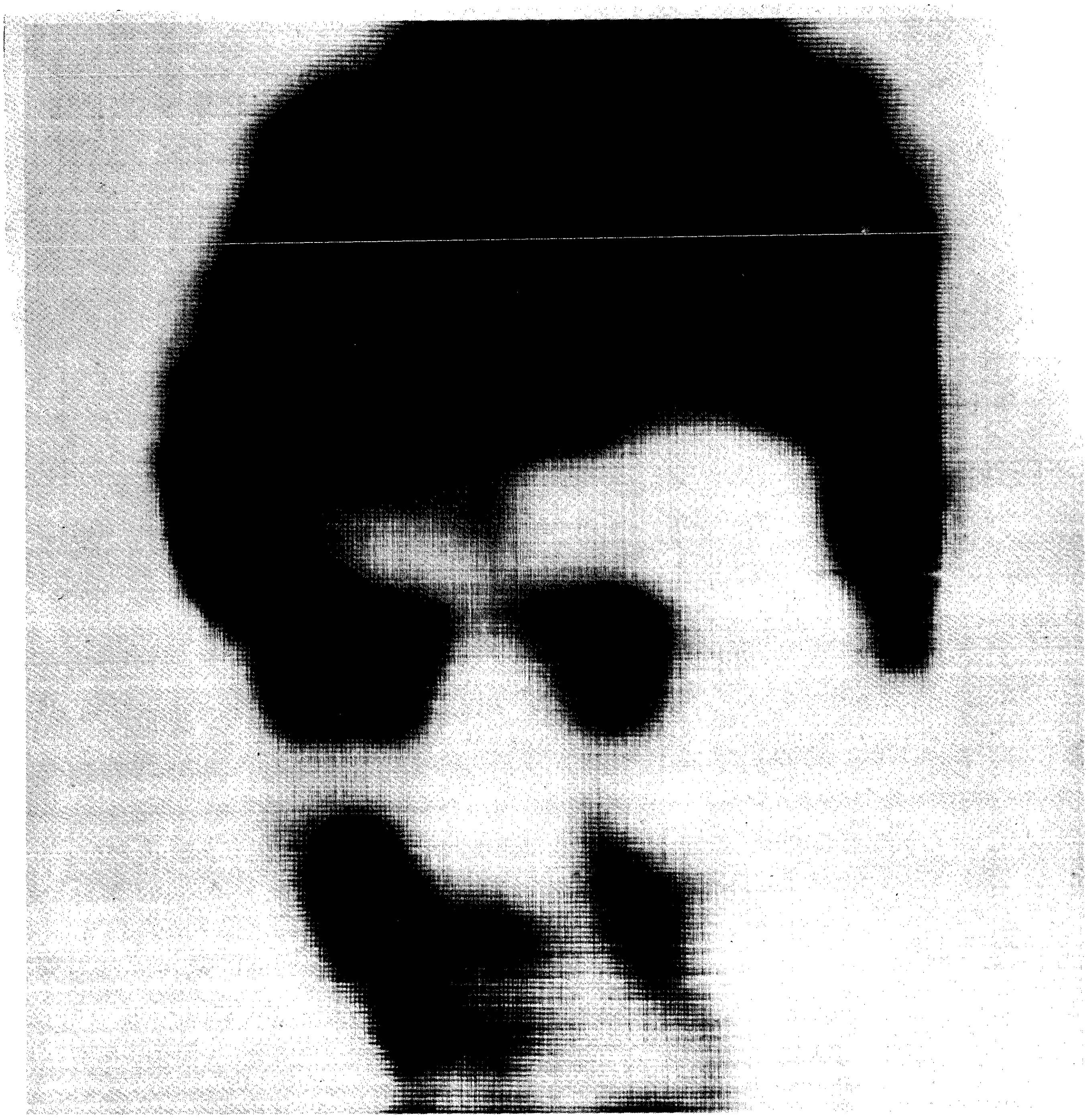


Figure 7—Linear spectrum quantization (5-bits)

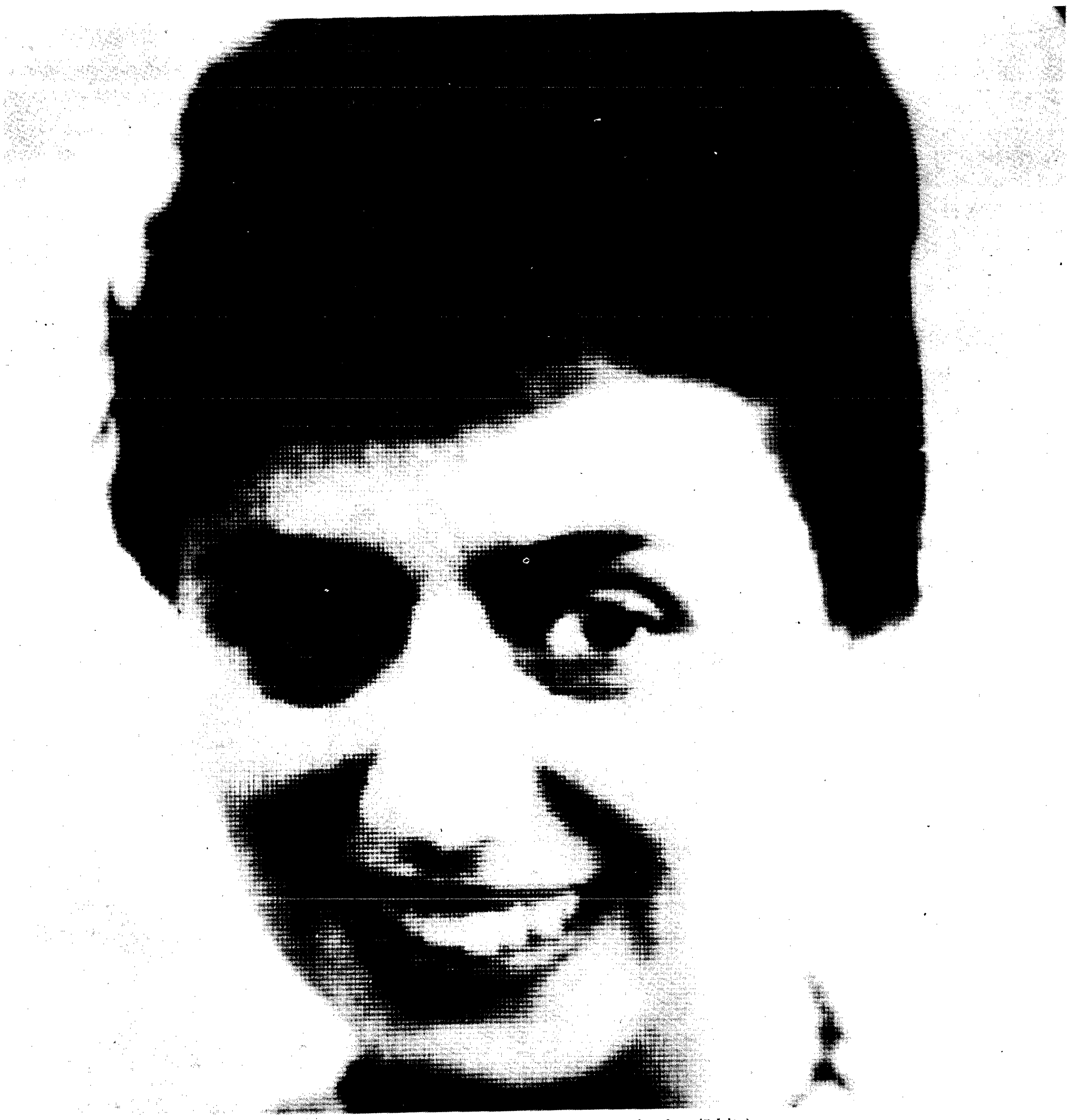


Figure 8—Nonlinear spectrum quantization (5-bits)

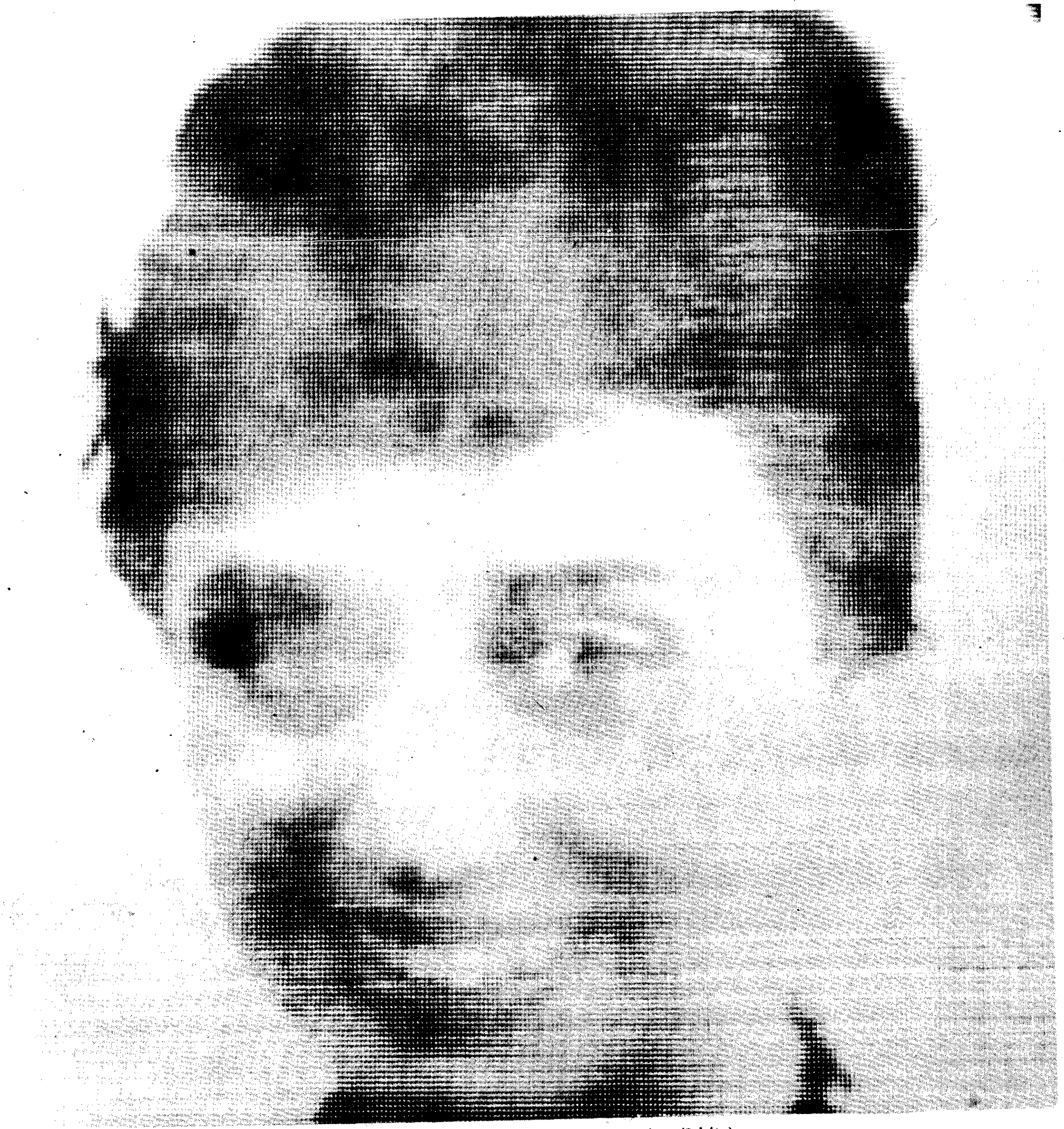


Figure 9—Nonlinear magnitude quantization (3-bits)

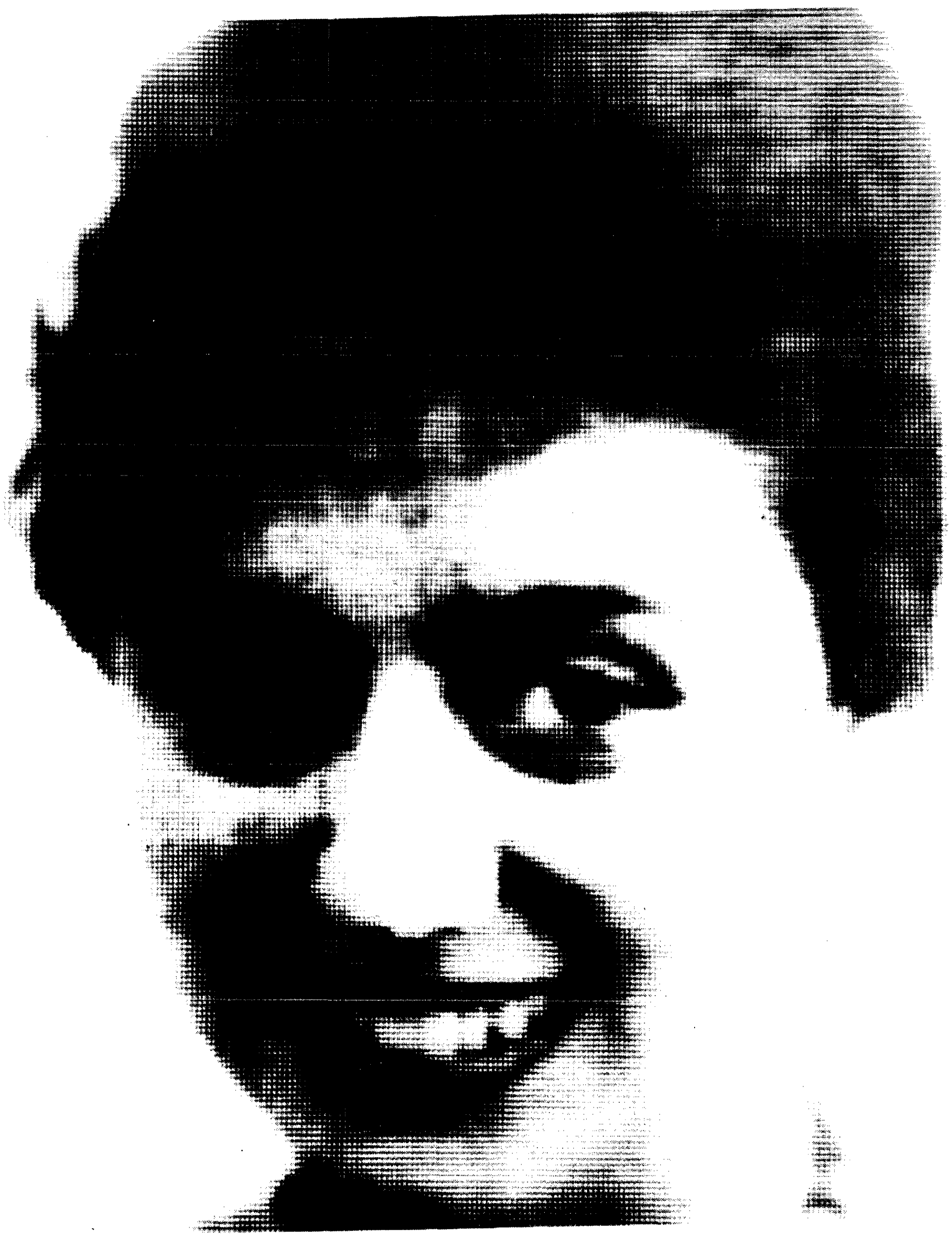


Figure 10—Uniform phase quantization (3-bits)

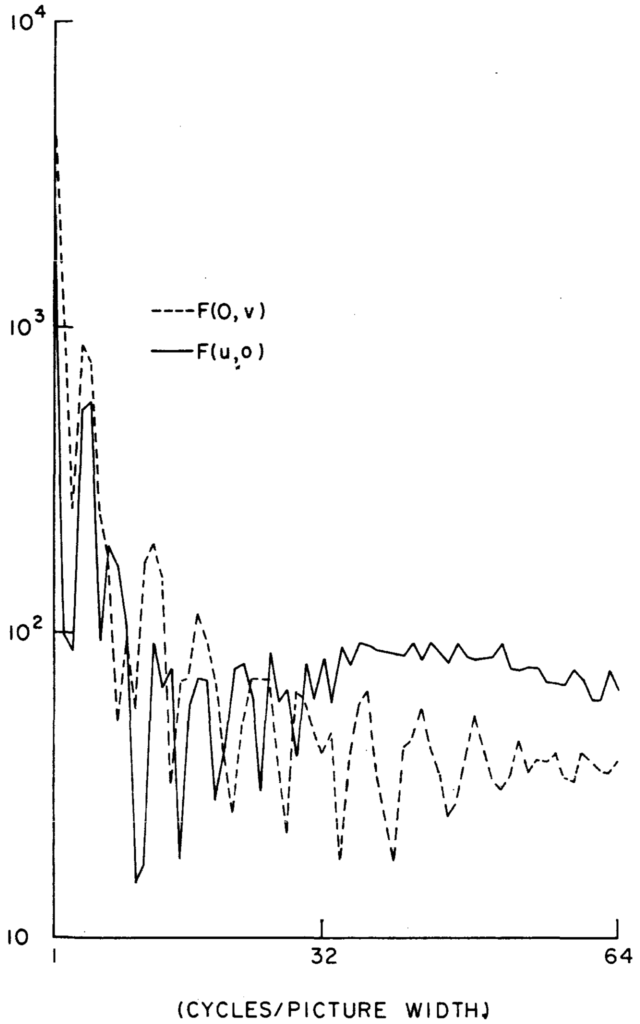


Figure 11—| F(u, o) | and | F(o, v) | for the test image

for the test image in Figure 11. Linear interpolation has been performed between data points to make the functions shown continuous.

Remarks

The theoretical predictions concerning additive, multiplicative, and phase noise are confirmed by the appearances of the images in Figures 3, 4, and 5. The image noise for the additive case (Figure 3) clearly has a white noise appearance. The appearances of multiplicative and phase noise were predicted by Equations (16) and (20). These equations stipulate that the power density spectra of image noise for the multiplicative and phase cases are identical, except for a constant, to the power density spectrum of F(u, v). The energy of F(u, v) (Figure 11) is primarily at low spatial frequencies. Therefore, the noise in the images of Figures 4 and 5 is primarily low-frequency noise.

Linear quantization of |F(u, v)| severely limits image resolution when an extremely small quantization interval is not used. Only low-frequency components are nonzero after linear frequency-domain quantization of the test image to 5-bits (Figure 7). Had Figure 11 been available before quantization, the resolution reduction could have been predicted. Figure 11 could also have been used to predict the improvement in mean-square error and image quality when logarithmic instead of linear quantization is used for |F(u, v)|.

REFERENCES

1 L J CUTRONA et al
Optical data processing and filtering systems
IRE Trans on Information Theory Vol IT-6 No 2
June 1960 386-400

2 J W COOLEY J W TUKEY
An algorithm for the machine calculation of complex Fourier series
Math Computation April 1965 297-301

3 W F SCHREIBER
The new scanner
Internal Memorandum Research Laboratory of Electronics
M I T Cambridge Mass April 1968 (unpublished)

4 A VANDER LUGT
Signal detection for complex spatial filtering
IEEE Trans on Information Theory Vol IT-10 No 2
April 1964 139-145

5 R B BROWN A W LOHMANN
Complex spatial filtering with binary masks
Appl Opt Vol 5 No 6 June 1966 967-970

6 W H LEE
Sampled Fraunhofer holograms generated by computer
Quarterly Progress Report No 88 Research Laboratory
of Electronics M I T Cambridge Massachusetts
January 15 1968 310-315

7 G B ANDERSON
Images and the fourier transform
S.M. Thesis Department of Electrical Engineering
Massachusetts Institute of Technology May 1967

8 W F SCHREIBER
Picture coding
Proc IEEE Vol 55 No 3 March 1967 320-330

Parametric description of a scan-display system *

by LAWRENCE A. DUNN, LAKSHMI N. GOYAL, BRUCE H. McCORMICK and VAL G. TARESKI

University of Illinois
Urbana, Illinois

INTRODUCTION

Automatic pattern recognition and graphical data processing have recently received considerable attention. In addition to analysis and processing of pictorial information, there is a need for interactive display systems to present both intermediate and final processed data. The subject of graphic display terminals has been extensively discussed in the literature. Scan-display systems oriented towards image processing, however, with particular attention to bypassing the central processing unit for as many tasks as possible have not had comparable development. This paper is focused on this latter area.

A computer system for image analysis and display has three principal constituents: image acquisition and display, image encoding for digital transmission, and finally procedures for classification of the encoded image. A new direction in image acquisition and display is to append a *Video Communications Net* to the central computer so as to provide the remote users with video transmission to centralized image processing facilities.

Requirements on image encoding for transmission of information are dependent upon application.[2] For example the bubble chamber data processing of high energy physics requires a very high positional resolution, although little demand is made on the gray-scale content of the picture. In the environmental sciences, however, gray-scale resolution is critical. Biomedical applications normally place stringent requirements on gray-scale resolution, but high positional resolution of the image is not required. Several systems are operative in high energy physics, such as PEPR[3] at M.I.T., CHLOE[4] and POLLY at Argonne National Laboratory, HPD[5] at Brookhaven National Laboratory, and HUMMINGBIRD[6] at Stanford Linear Accelerator Center. Other systems are FIDAC[7] in biomedicine and KARLSRUHE[8] in automatic photointerpretation. However, we feel that these systems are unnecessarily specialized, and there is need for a more versatile and general system applicable to diverse disciplines on an integrated basis and amenable to modification as the need arises. In this paper we define an integrated system for the image acquisition and display.

Figure 1 shows the proposed system. Video switching matrix provides the facilities for remote users. High resolution CCTV cameras are provided for image encoding and acquisition. Remote video consoles, consisting of two high resolution monitors and a teletype set, provide information display at the remote user's end. Videograph printer outputs a facsimile copy at video rates, where the copy can have any admixture of text, graph or half-tone pictures. Microimage store provides the system with an extensive store of images—as direct images and not as digital data—and finds application in information retrieval areas such as library automation and biomedicine, where there is need for a permanent huge mass storage. High resolution scanners allow the scanning of the film for accurate measurement purposes and also allow the construction of images on film. High resolution monitors are slaved to the scanner system in a manner which allows the monitors to borrow inexpensively the scanner control. If video scan converters are part of the video switching matrix, video images can be treated as if they were on film and thus the same encoding techniques and the same programs could be exploited for both without

* Supported by Contract AT(11-1)-1018 with the U.S. Atomic Energy Commission and by the Advanced Research Projects Agency.

Based on an internal document published in the Department of Computer Science.[1]

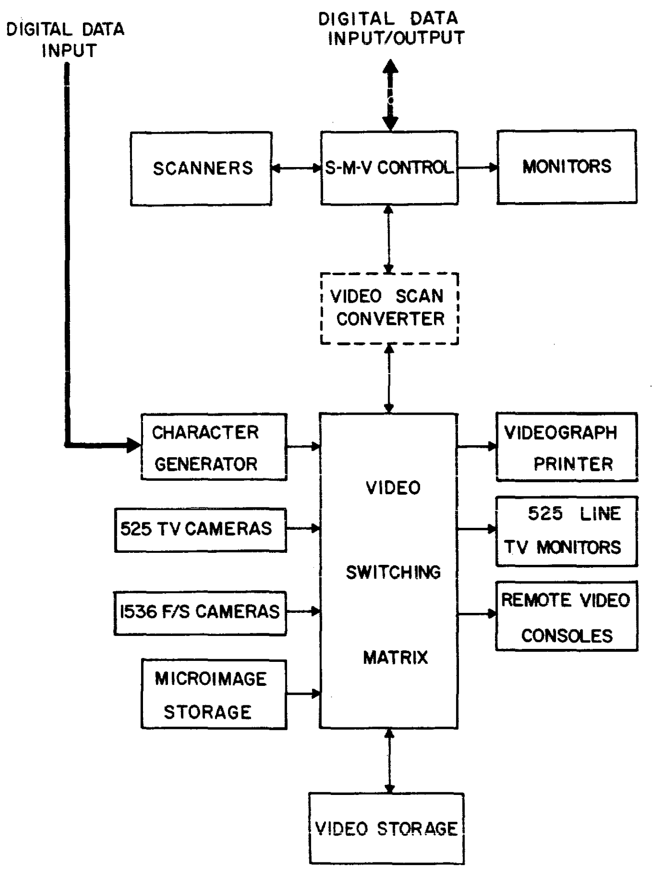


Figure 1—Block diagram of the scan/display system. Note that the video scan converter can be bypassed under program control

program change. However, the video scan converter is not essential, since the controller can handle constraints of the video system with some sacrifice in resolution. Character generator provides facilities for message handling and display in general, particularly for CAI, which along with the scanner provides the flexibility of displaying line drawings and half tone pictures intermixed with the text. More details about the devices shown in Figure 1 are given in Appendix A.

Emphasis in the paper is given to inter-media transformation options—translation, magnification and rotation—that can be effected by control of position counters, to constraints necessary for maintaining media compatibility, and to alternate digital representations of an image. The major goal of the paper is to abstract the system parameters and to develop the relations among them. These system parameters are listed with their definitions in Appendix B.

Design values of these parameters for the Illiac III Scan-Display System are tabulated in Appendix C. For this system the format of the scan/display parameters is shown in Figures 18 and 19.

Ways of digitally representing an image

By selecting a grid and associated coordinate system an image can be represented as a digital string of encoded local samples. The coordinate system allows specification of a sampling position, or by implication a sampling sequence; and the grid mesh allows specification of a sampling resolution, or by implication a sampling frequency. One conventional image digitization procedure specifies sequential sampling along successive lines parallel to a specified coordinate axis. To describe these linear sweeps we will use the terms 'scan line' and 'scan axis.' Although several string definitions are possible each will always contain, either explicitly or implicitly, two correlated types of information:

the coordinates of a point, and

the image density level in some localized area about the point.

If the localized sampling area is not circular in shape then a third type of information can be included:

local orientation of the sampling area.

The encoding/decoding format for these three items (position, density and orientation) is invariant in any given string definition. Any further interpretation of the string requires knowledge of the sampling technique employed.

Specifying a sampling sequence, a digital string representation can be formed in two different ways:

(r) encoding the image density level for each coordinate in the sequence, or

(c) forming an ordered substring of coordinates by selecting only those from the given sequence for which the image density level satisfies some prescribed criteria.

For a particular image each of the two strings is unique within the reproducibility limits of density detection and encoding.

String (r) is usually more representative and economical (in terms of string length) for images with a frequently varying density, i.e., with large amounts of detailed information—as a page of text with illustrations or a stained tissue section.

String (c) is more economical and representative for sparse images such as an engineering drawing or bubble chamber negative. For these images high resolution is needed along the scan axis but the scan lines can be relatively far apart without loss of information. A rectangular or slit-like sampling area is effectively employed if its orientation is within approximately 45° of being perpendicular to the scan line. However, to sample all possible orientations it is necessary to:

(c1) sample each scan line several times—once for each of a sequence of orientations,

(c2) sample a second sequence in which scan lines are perpendicular to those of the first.

String (c) can be a function of the sampling technique, i.e., one can sample *for* and recognize *only* a preselected class of features. Two possible classes are:

(c3) silhouettes, or images containing large areas of uniform density—adjacent areas are distinguished by a sharp discontinuity in image density. Chromosome karyotyping falls in this class.

(c4) outlines, or sparse line drawings—these are a special case of silhouettes. The engineering drawings and bubble chamber negatives are in this class.

Sampling criteria corresponding to each of these classes can be stated as:

(c3a) select the point if the *change* in image density between it and the previous point(s) exceeds some threshold.

(c4a) select the point if the image density *increases above and then decreases below* some threshold between it and the previous point(s).

The image density level in each case can be recorded in the string representation along with the coordinates.

If it is known that the image contains line (or boundary) information—perhaps from having processed a string obtained in one of the aforementioned ways—then it may be useful to track the constituent lines by sampling a sequence of localized areas with restricted sets of orientations. This tracking procedure then generates a sequence of head-to-tail vectors or, in the limiting case, a string of incremental displacements. This concept is also useful in creating an image by first constructing the incremental string. A natural interpretation of the string is:

(i) an ordered set of commands defining starting point, line width, line intensity and the (incremental) segment vectors.

Curved lines obviously can be represented as a sequence of sufficiently short segmental displacements.

These three ways of forming strings—(r), (c) and (i) are correspondingly defined as the *Raster*, *Coordinate*, and *Incremental* data formats.

Raster format

Stated in PL/1 the sampling algorithm for an X-axis scan including the optional orientation sampling is:

```
  IF¬SLIT then ΘB = ΘE;
SY: DO  Y = YB BY ΔY TO YE;
SΘ: DO  Θ = ΘB BY ΔΘ TO ΘE;
SX: DO  X = XB BY ΔX TO XE;
        'encode/decode G(Y, Θ, X)';
        END SX;
        END SΘ;
        END SY;
```

where (XB, YB) and (XE, YE) define a rectangular area to be sampled at increments of (ΔX, ΔY). Each scan line is sampled once for each orientation from ΘB to ΘE at increments of ΔΘ.

The total number of samples is:

N_s = Number of lines × number of orientations per line × number of samples per orientation

and the number of bits required for storage is:

N_b = Number of bits per sample × number of samples.

Coordinate format

Stated in PL/1 the sampling algorithm for an X-axis scan is:

ENCODING (READ)

```
  IF¬SLIT THEN ΘB = ΘE;
SY: DO  Y = YB BY ΔY TO YE;
SΘ: DO  Θ = ΘB BY ΔΘ TO ΘE;
SX: DO  X = XB BY ΔX TO XE;
  IF 'Criteria satisfied' THEN
CYES: DO;
        IF 'First time for this Y and Θ' THEN
        FYES: DO; 'output Y';
```

```
                IF SLIT THEN 'output Θ';
                END FYES;
            FNO: DO; 'output X';
                IF NGL >2 THEN 'output G';
                END FNO;
            END CYES;
    CNO:    END SX;
            END SΘ;
            END SY;
```

DECODING (WRITE)

```
    INIT:   'input YIN';
            IF SLIT THEN 'input ΘIN';
            'input XIN';
            IF NGL >2 THEN 'input G';
    SY:     DO  Y = YB BY ΔY TO YE;
    SETΘ:Θ = ΘIN;
            IF Y = YIN THEN
            SX: DO X = XB BY ΔX TO XE;
                IF X = XIN THEN
            MOD: DO; 'modulate beam';
                'input next coordinate';
                IF 'coordinate is a new Y' THEN
            NEWY:DO; IF SLIT THEN 'input ΘIN';
                'input XIN';
                IF NGL >2 THEN 'input G';
                GO TO SETΘ;
                END NEWY;
            NOTNEWY: IF NGL >2 THEN 'input G';
                END MOD;
                END SX;
                END SY;
```

where (XB, YB) and (XE, YE) define a rectangular area to be scanned at increments of (ΔX, ΔY). In encoding each scan line is swept once for each of the orientations between ΘB and ΘE, where ΔΘ is the orientation increment. Actual encoding takes place only when the criteria is satisfied. In decoding, only those scan lines which are specifically read in are swept at the input orientation, and writing takes place (or more generally, is initiated) only at those positions which match the input coordinates.

NGL is the number of gray levels. If this number is greater than two, additional encoding/decoding is necessary to obtain the gray levels. For NGL = 2, the fact that the criteria is satisfied implies the gray scale information.

The most general coordinate string of an X-axis scan has the form:

Y coordinate, Θ, X coordinate, gray scale, X coordinate, gray scale, X coordinate, gray scale. . .

Y coordinate, Θ, X coordinate, gray scale, . . .

Increment format

The incremental string is composed of a sequence of elements that can be interpreted either as a segment vector or as an incremental command. The segment vector is composed of two incremental displacements, DX and DY, the corresponding signs for each displacement, SX and SY, and the 'beam condition.' DX specifies the number of unit cells the beam is to be displaced in the X direction and DY specifies the number of unit cells the beam is to be displaced in the Y direction. The beam condition is given as being either on or off during the move.

If (DX, DY) = (0, 0), the two signs, (SX, SY), and the 'beam condition' are interpreted as an incremental command, where incremental commands have the following semantics:

H	*H*alt, close out the operation.
IT	*IT*erate (magnify) the next segment vector. The next two elements in the string should be a count followed by a segment vector.
MB	*M*odulate the *B*eam intensity (at a fixed position).
NOP	*N*o *OP*eration
RGR	*R*eset *G*rid *R*esolution
RSO	*R*eset *S*tencil *O*rientation
RSS	*R*eset *S*tencil *S*ize and/or shape
RVB	*R*eset *V*ector *B*egin point

The IT command with its count is equivalent to having the same segment vector appear sequentially in the string by the number of times given in the count byte. If the displacement is (0, 0) the IT command is ignored.

The four commands that reset parameter values are followed by string elements containing the new parameter values. The element format is the same as that defined for the initializing parameter string.

For the incremental format (XD, YD) and (XE, YE) are interpreted as defining a file area, outside of which no recording (film) or displaying will be allowed to take place. (XB, YB) becomes the initial point from which the beam will start the first segment vector.

Figure 2 shows the use of incremental vectors with the command RSS inserted at point P (between vectors (2,2) and (3,1)) and with command RGR inserted at point Q.

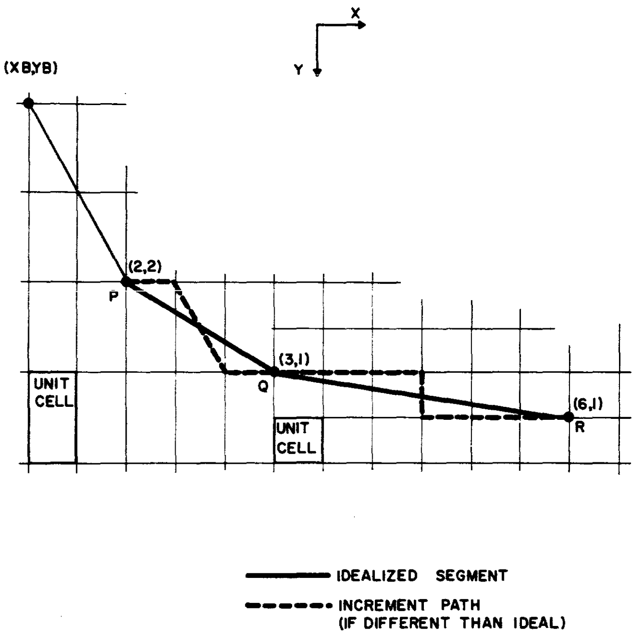


Figure 2—Segment vector plotting. Note change in stencil size at P and change in resolution (unit cell) at Q

Resolution, sampling and scanning parameters

Resolution

In order to encode a digital representation of an image it is necessary to impose a grid and coordinate system upon it. It is convenient to conceive the *total image or raster area as a unit square* and to interpret the addressable positions of the image as fractional coordinates ranging between 0 and 1. The mesh of the grid superimposed on this range is then 2^{b_g} where b_g is the number of bits used to specify coordinate position. The smallest resolvable square has sides of 2^{-b_g} *units* and is termed a *gross basic cell.*

The aspect ratio of the raster area need not be 1 : 1. The physical interpretation of the basic cell will more generally be a rectangle, hence the effective resolution along one axis may differ from that along the other.

The adopted coordinate system—left-handed rectangular—is a natural one for most textual material and also corresponds to standard video practice of left-to-right top-to-bottom scanning.

In concept one achieves the physical limit of resolution by choosing b_g sufficiently large. In practice this is difficult to implement and one distinguishes between a gross position counter specified by b_g, and a vernier position counter specified by b_v. The vernier counter has a sign bit, s_v, and is interpreted as a signed position relative to the gross position. This is equivalent to overlaying a vernier grid in the immediate vicinity of a gross coordinate position (which can be interpreted as a local 'benchmark'). One then has a localized grid of mesh $2^{b_g+b_r}$ where b_r is the number of bits in the vernier counter used to extend the gross resolution. The smallest resolvable square has sides of $2^{-(b_g+b_r)}$ *units* and is termed a *vernier basic cell.* The remaining b_o bits in the vernier counter define the gross-vernier overlap, or the *maximum vernier window* as having sides of 2^{b_o} *gross basic cells* (see Figure 3) or $2^{(b_o-b_g)}$ units.

The above discussion defines the design parameters that determine maximum position resolution. Not all applications warrant this maximum resolution. More importantly one cannot contrive efficient scanning-recognition algorithms without a range of resolution options. One clearly wants independent choices of resolution for the two axes, either because the application warrants it, or becuase the format warrants it, as in the case when scanning in the coordinate format using a slit-like sampling area.

The parameters p and q specify the sampling resolution as every 2^p basic cells in the X-direction and every 2^q basic cells in the Y-direction:

$$\Delta X = 2^p \text{ basic cells} = 2^{p(b_g+k)} \text{ units} \quad (1)$$

$$\Delta Y = 2^q \text{ basic cells} = 2^{q(b_g+k)} \text{ units} \quad (2)$$

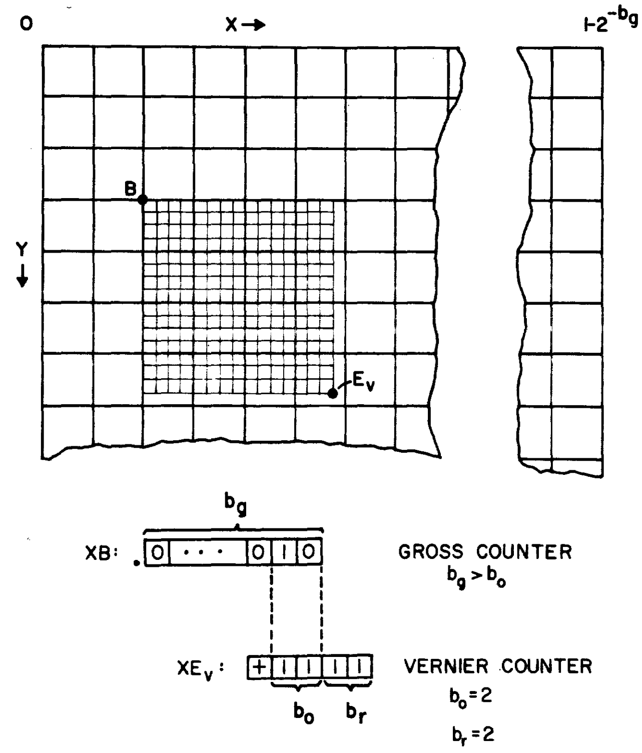


Figure 3—Maximum vernier window area with respect to B coordinate. Shown is an overlap of two bits (b_o) and maximum local resolution for a resolution extension of two bits (b_r)

The gross/vernier selection determines k as $0/b_r$. The smallest resolvable rectangle is ΔX *by* ΔY units and is termed the *unit cell.*

The position counter is incremented at the p^th or q^th significant position of the gross/vernier counter, hence the number of significant position bits is:

	X	*Y*
raster gross:	$b_g - p$	$b_g - q$
raster vernier:	$b_g + b_r - p$	$b_g + b_r - q$

For the coordinate representation it is desirable to increase gross sampling resolution along the scan axis while incrementing the position counter at the p^th significant position as described above. This can be done by interpolating between counter increments and concatenating the b_c interpolation bits to the $b_g - p$ significant counter bits. One then has for the coordinate resolution:

coordinate gross: $b_g + b_c - p$ significant bits.

A natural choice is to make $b_c = b_r$ since b_r reflects the physical limit of resolution.

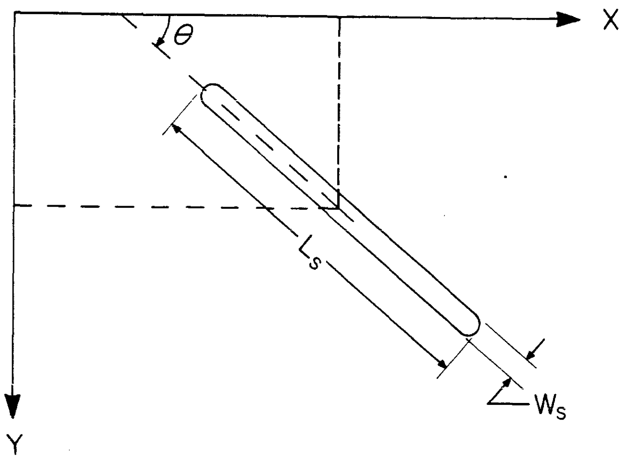
Sample encoding

The maximum number of image density states for a read command or recording beam intensity states for a write command are indicated by the gray scale parameter, n. The number of encoded bits is interpreted as 2^n, hence the number of possible states as 2^{2^n}. The maximum value of n is specified as n_{max}.

Triggering or filtering of output information may be done by either a standard level discriminator or by a specially designed plug-in unit. The value assigned to the parameter T chooses between these two.

The parameter B_s will distinguish between the choice of a standard size spot ($\simeq$1 gross basic cell in diameter), and a slit or non-standard spot size. If the non-standard option is taken, then the parameters u and v define the sample width and length as 4^u and 2^v gross basic cells respectively. The range of circular spot sizes can be specified by u with $v = 0$.

If the sample area is slit-like then the orientation becomes significant. The *unit of angle* is the circle, or radians/2π. The angular resolution is 2^{-b_a} units where b_a is the number of bits used to specify an angle. The angular sweep Begin-End coordinates, (θB, θE), specify the range over which incrementing is to be accomplished. The increment is defined by the parameter z as $\theta = 2^{z-b_a}$ units. The slit is swept the length of a scan line for each value of θ in the specified range.



SLIT:	SPOT:
$W_s < L_s$,	L_s = MINIMUM
	W_s = DIAMETER
θ RELEVANT	θ IRRELEVANT

Figure 4—Slit/Spot geometry

Figure 4 illustrates the geometrical definition and angular reference.

Scanning rate

Sweep velocity will in general be limited by the following:

1. positional digital-to-analog response time
2. maximum channel data transfer rate
3. number of bits of gray scale encoding/decoding

Sweep velocity can generally be expressed as a function of the following parameters (see Appendix B), where a constant clock rate for incrementing the positional counter is assumed:

$$V_s = c_1 \cdot f(p,q,A) \cdot g(DF, K, n, n_{max}).$$

Here f is determined by the scan axis and unit cell selection:

$$f(p,q,A) = (1-A) \cdot 2^p \cdot \delta x + A \cdot 2^q \cdot \delta y.$$

The maximum *continuous* data rate is required for the Raster format with $n = n_{max}$, where g can be normalized as

$$g = 2^{-n_{max}}.$$

The data per sample are higher for the coordinate format, but the average data rate is normally less than for Raster. (Otherwise the image would be more optimally encoded in a raster format.) Buffering is needed to handle local bursts of perhaps three or four consecutive samples. The amount of data for coordinate representation are essentially (though not absolutely) independent of the number of bits of gray scale encoding/decoding.

For an X-axis scan at *constant sample rate* one then has

$$V_s = c_1 \cdot 2^{-n_{\max}} \cdot 2^p \cdot \delta x.$$

c_1 can then be determined from the clock frequency. This equation yields a constant sample rate with the Raster format for any value of n.

Since the data per sample is 2^n bits, one obtains a *constant data rate* by introducing the quantity $n_{\max} - n$ in the exponent yielding:

$$V_s = c_1 \cdot 2^{-n_{\max}} \cdot 2^{p+(n_{\max}-n)} \cdot \delta x.$$

Here instead of incrementing the position counter at the pth significant position, the burden on the positional D/A conversion is appreciably lessened if one increments only at the $[p + (n_{\max} - n)]$th significant position and interpolates to obtain samples at the $(2^{n_{\max}-n} - 1)$ positions between counter increments. The constant data/sample rate option for a given resolution is then determined by the parameter K.

Window area

The term *window* was introduced above in defining the maximum area that can be covered in a single operation with vernier resolution. Two pairs of coordinates serve to specify the portion of an image to be scanned, encoded and displayed. They are termed the Begin coordinates (XB, YB), and End coordinates (XE, YE), and are interpreted as defining diagonally opposed corners of a rectangle. Since the position counters may be decremented as well as incremented any one of four B-E combinations may be chosen to specify the same window area. This feature and the coupled-uncoupled option of the monitor position counters allow the Rotation Group transformations described below. The coordinates must be multiples of (ΔX, ΔY).

Scan format (lattice and sequence)

The scan format, determined by the parameters *L*, *S* and *A*, imposes a *Lattice* upon the grid in terms of the *unit cell* and specifies the scan line sampling sequence. The scan axis is specified by A as X or Y, and S selects the scan line sequence as interlaced or sequential. These two parameters along with the B-E combination completely determine the sampling sequence. One obviously starts with the Begin coordinate and terminates with the End coordinate, determining the direction of sampling along a scan line. The hexagonal/rectangular lattice option is defined by L as described below.

The matrices in Figures 5 and 6 illustrate the sampling sequences for a scan direction parallel to the X-axis on a *Rectangular Lattice.* The position of point No. 1 in the lattices is specified by the *Begin Coordinates.* The *End Coordinates* specify point No. 42 in the *Sequential* case (Figure 5) and point No. 15 in the *Interlaced* (Figure 6) case. Since the Begin and End coordinates are constrained to be multiples of ΔX and ΔY they will always be member points of the Lattice.

Figure 7 illustrates the relative positioning of points and their sampling sequence for the two scan directions on *Hexagonal Lattices* when line sampling is *sequential.* The Hexagonal Lattice is obtained by shifting the lattice points of *odd* numbered scan lines in the corresponding *Rectangular Format* by an amount $\Delta X/2$ (or $\Delta Y/2$) along the positive scan axis. Otherwise hexagonal formats are analogous to their rectangular counterparts.

The *Interlace* option and the adopted coordinate system are particularly suited for communication with a standard video network through an analog-to-digital interface. Figure 6 shows the relation between the

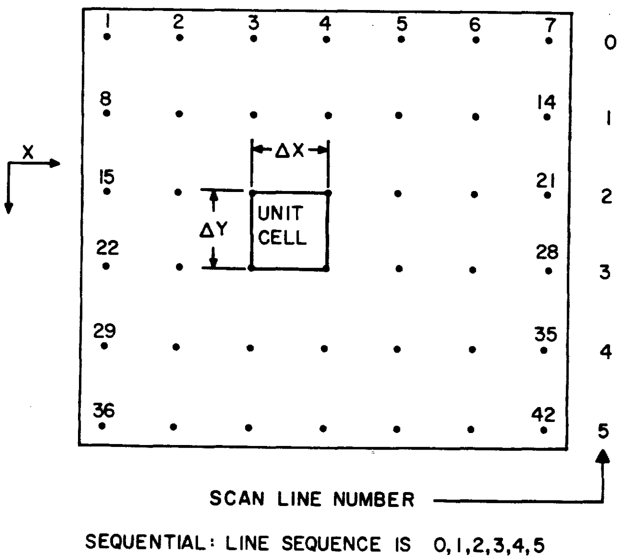


Figure 5—Sequential scan along X-axis using a rectangular lattice. ($\Delta Y = \Delta X$)

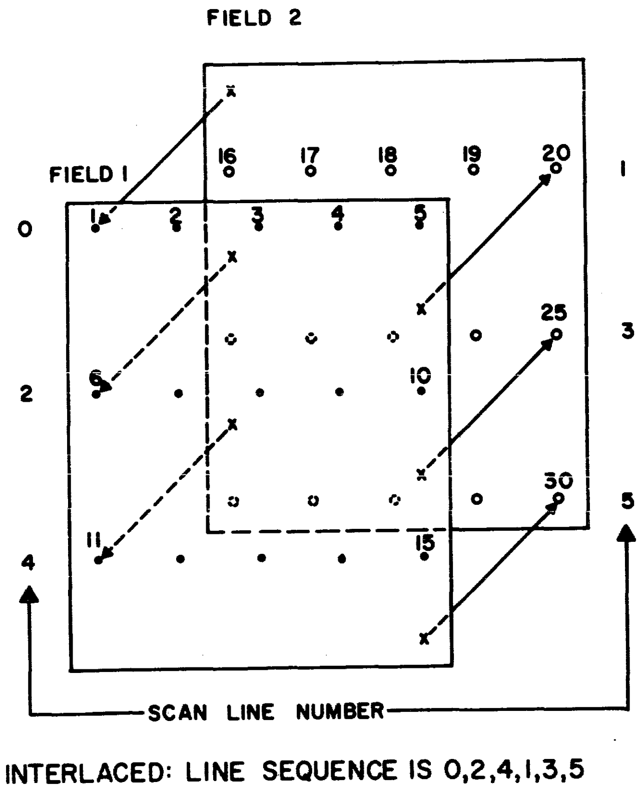


Figure 6—Interlaced scan along X-axis using a rectangular lattice. ($\Delta Y = \Delta X$)

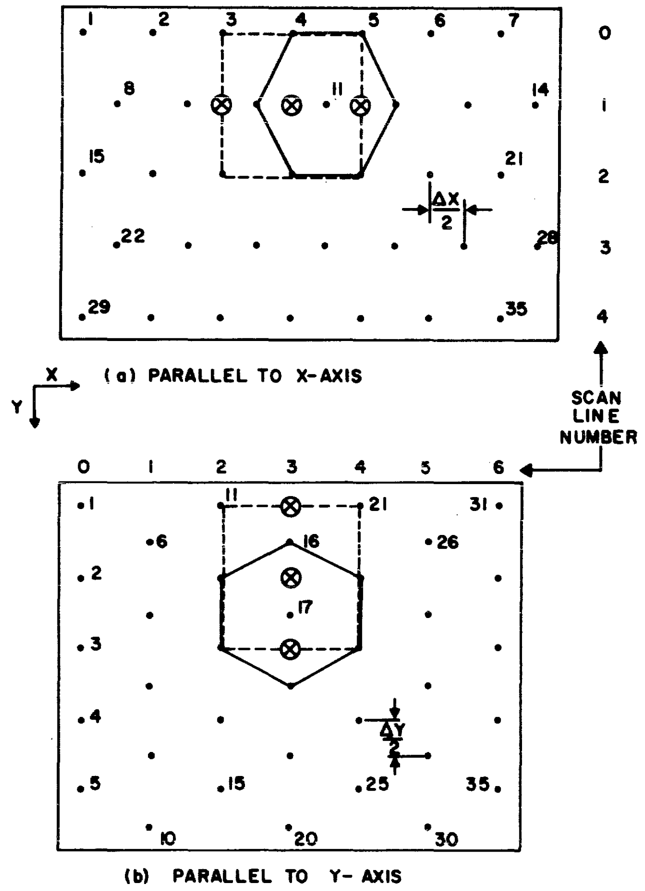


Figure 7—Sequential scan along (a) X-axis, and (b) Y-axis using a hexagonal lattice. The hexagons around point 11 in (a) and point 17 in (b) illustrate neighbor points. The dotted rectangles show neighbor points in the corresponding rectangular lattice. ($\Delta Y = \Delta X$)

fields and the sample points for the interlace option.

Lines have been drawn between points surrounding point No. 11 in Figures 7(a) and 8 and point No. 17 in Figure 7(b) to illustrate the concept of neighbor points. An interior point of the Hexagonal Lattice has *6* neighbor points whereas that of the Rectangular has 8. Since the Hexagonal Lattice is not regular (it is rhombic), although it is nearly so for $\Delta X = \Delta Y$ (see Figure 7), neighbor points are *not* all equidistant from their interior point; but they always partition by distance into two sets of 4 and 2 points each. Those for the Rectangular Lattice partition generally into three sets of 4, 2 and 2 points each, and for $\Delta X = \Delta Y$ the two sets of 2 and 2 become a single set of 4. Compare Figure 8 with Figure 7.

Video compatibility

When interfaced by a video scan converter the video network can be handled as a scanner. However, the scan converter is not essential since the S-M-V Controller can be designed to satisfy the constraints of the video systems. The following discussion uses the parameter i to identify the video system within the network;

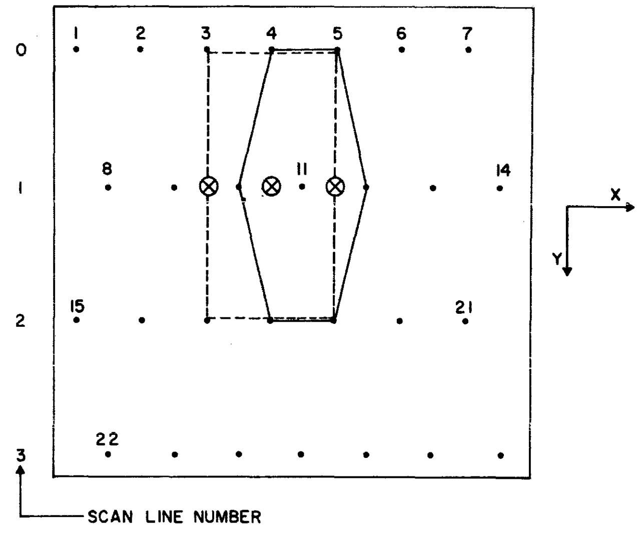


Figure 8—Same as Figure 7 (a) with $\Delta Y = 2\Delta X$

it assumes the controller-video link to be direct and develops the corresponding constraints.

Since the viedo line sweep time, $\Delta t(i)$, is a fixed parameter the number of data bits that can be transferred to or from core per full scan line is constrained by the I/O channel capacity, where the maximum bit transfer rate is f_b. A buffer will allow a 'burst-mode' sampling for some fractional part of the scan line—hence higher resolution in sampling a vertical band can be achieved.

The Incremental string cannot be passed directly to video; it must first be passed to a scan converter.

The Coordinate string with video has a resolution along the scan line equivalent to Raster resolution with $n = n_{max}$, hence the scan axis counter increment, ΔC, is given by:

Raster with constant data rate option:

$$\Delta C = 2^{p+(n_{max}-n)}$$

Raster with constant sample rate and Coordinate:

$$\Delta C = 2^p.$$

One would normally choose the constant data rate option. The Coordinate string resolution can only be locally maintained because the buffer storage limits the number of successive samples in contiguous unit cells. This is not a severe restriction since the coordinate representation is not very meaningful unless the image is rather sparse. Note that orientation sampling is meaningless with video.

The entire video discussion is from the point of view of the Raster string representation. The string will have existence only when core memory participates; otherwise there is only the analog signal transfer between the other three media. The resolution results are valid independently of the point of view.

Horizontal (X-axis) resolution

To achieve a maximal uniform resolution across a full scan line it is necessary to maintain a constant data rate. We choose the largest j such that

$$2^j \leq f_b \,.\, \Delta t(i) \qquad (3)$$

and hence maximize and fix the number of bits in a full line of sampling. Position resolution and gray scale resolution are not independent: the number of bits per sample is 2^n, hence equation (1) constrains the number of samples in a full line to be

$$S(j,n) = 2^{j-n}\,. \qquad (4)$$

Maximizing position resolution minimizes gray scale resolution and vice versa. S(j,n) achieves its maximum value for n = 0:

$$S_{max}\,(j) = S(j,0) = 2^j \qquad (5)$$

Table I shows values of $S_{max}(j)$ for three different video systems. Parameter values used in the calculation are given in Appendix C. If a window contains only a fractional part of a scan line, then only a corresponding fractional part of S(j,n) samples can be obtained along each scan line.

Sampling at the position counter stepping frequency, f_c, maintains the aforementioned data rate for an $2^{n_{max}}$ bit gray scale datum. An interpolation counter is used to achieve the rate for other choices of image density resolution with the constant data rate option:

$$\Delta C = 2^{p+(n_{max}-n)}.$$

A 1-1 correspondence between a full scan line at video and the S-M-V control grid requires:

$$S(j,n) \cdot \Delta X = 2^{b_g} \text{ basic cells.}$$

Substituting for S(j,n) from equation (4) yields:

$$(\Delta X)_{max}\,(j,n) = 2^{b_g-(j-n)}, \text{ hence}$$

$$p_{max}(j,n) = b_g-(j\text{-}n). \qquad (6)$$

This defines the achievable video resolution along

Table I—Inherent characteristics of three video systems and maximum sampling resolution, $S_{max}(j)$, as constrained by other media

i	$N(i)$	$\Delta t(i)$ μ-sec	$S_{max}(j)$	j
1	525	≈ 56	512	9
2	1536S	≈520	4096	12
3	1536F	≈ 43	256	8

F = FAST SCAN
S = SLOW SCAN

Table II—Maximum sampling resolution and interdependence of position and gray scale resolutions as constrained by media characteristics. The main entry is the maximum number of samples per video scan line, S(j, n), and the value in parentheses is the corresponding maximum value of ΔX, $(\Delta X)_{max}(j,n)$

j	n = 0	n = 1	n = 2	n = 3	g(j)
9	512 (8)	256 (16)	128 (32)	64 (64)	0
12	4096 (1)	2048. (2)	1024 (4)	512 (8)	-1
8	256 (16)	128 (32)	64 (64)	32 (128)	3
2^n, bits of gray scale	1	2	4	8	

scan lines as well as the largest useable sampling increment that may be associated with the S-M-V control grid scan axis. Table II illustrates the interdependence of position and gray scale resolution for the three video systems of Table I. It contains the values of S(j,n) as constrained by equation (4), and in parentheses the corresponding maximum values of ΔX as given by equation (6). All parameter values used are listed in Appendix C. The function g(j) is explained in the following section.

Vertical (Y-axis) resolution

Having determined the video resolution along a scan line we now determine the vertical sampling so as to achieve a unit cell match to the S-M-V control grid. Assuming an X-axis scan at the controller, this constraint requires the video line sampling frequency to be

$$(\Delta Y)_v = \frac{r_v}{r_s} \cdot \frac{N(i)}{S(j,n)} \cdot \frac{\Delta Y}{\Delta X}, \qquad (7)$$

where r_s and r_v are the aspect ratios at the control grid and video, respectively. N(i) is the number of lines per video frame. The unit cell resolution ratio at the S-M-V control grid is ΔX/ΔY, and the corresponding video resolution ratio is $1/S(j,n)\ N(i)/(\Delta Y)_v$. Using equations (1), (2) and (4) equation (7) can be written as

$$(\Delta Y)_v = r_v/r_s \cdot N(i) \cdot 2^{q+n-p-j},$$

which may be restated as:

$$(\Delta Y)_v = f(j) \cdot 2^{q+n-p},$$

where

$$f(j) = r_v/r_s \cdot N(i) \cdot 2^{-j}.$$

As in the discussion of horizontal resolution we wish to approximate by powers of 2, and determine a g(j) such that

$$f(j) \cong 2^{g(j)}.$$

The video line sampling frequency is then determined at the S-M-V controller from

$$(\Delta Y)_v = 2^{q+n+g(j)-p}$$

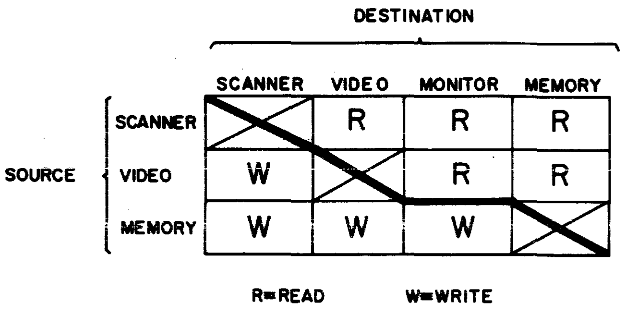
since the four terms in the exponent are defined by the parameter assignments.

Media selection

Media selection is determined by the parameters F, V and C. The choice of a Read or Write command is then determined from the source-destination matrix shown in Figure 9. Of the sixteen possible states for F, V, C and Read/Write ten are allowed as meaningful or useful, and this may be succinctly stated as:

States	*Source*	*Destination*	*Command*	*Figure*
4	$(F \oplus V)$.	$(C \oplus C)$.	READ	15
1	F .	V.C .	READ	16
1	V .	F.C .	WRITE	16
4	C .	$(F \oplus F) \cdot (V \oplus V)$.	WRITE	17

where $\oplus$ means exclusive OR. As indicated these states are illustrated in Figures 15-17. A destination medium always exists since the monitor participates in all



		DESTINATION			
		SCANNER	VIDEO	MONITOR	MEMORY
SOURCE	SCANNER		R	R	R
	VIDEO	W		R	R
	MEMORY	W	W	W	

R=READ W=WRITE

Figure 9—Command matrix. Read/Write selection as a function of selected media and desired transfer direction

operations. Whenever core memory (C) is *not* the image source, display at the monitor or video can be indefinitely repeated by setting the Regeneration parameter, J.

When transferring an image between two media, it is necessary to consider:

a. the X/Y aspect ratios, and
b. the X/Y resolution ratios.

If either of these ratios differ, then a contraction quite independent of any magnification can take place along one of the axes. Both sources of image distortion may be averted by matching *aspect ratios of the unit cell* at source and destination media. Since several media may be involved it is useful to adopt the concept of an S-M-V control grid and coordinate system through which any inter-media transfer must pass, as illustrated in Figures 14 through 17. One then forces a match between each medium and the S-M-V control grid. All position and resolution specifications in the parameters can be interpreted as referring to the S-M-V control grid and coordinate system. They must be specified with a particular medium (or media) in mind, however, and must be compatible with its associated characteristics. Scanner and monitor are completely dominated by the S-M-V Controller, hence the unit cell match is easily accomplished. The same is true of core memory as a destination medium. As a source medium the core memory unit cell match is under program control, and it is therefore necessary to associate inviolate unit cell as well as other parameter information with any string representation. Except for initiating a video scan, the link between S-M-V Control and video is basically an information transfer link. The video match is effectively accomplished by selectively transferring information from video (say every other scan line) and by holding back information to video (say blanking every other scan line) by employing $(\Delta Y)_v$ as discussed in the section on Vertical Resolution.

Extending this "match-to-control" concept allows all the transformations (rotation, magnification and translation) discussed in the following sections as well as the dual interpretations of video as a source medium.

The entire transformation discussion is from the point of view of gross resolution. Vernier resolution differs basically in having an inherent magnification of 2^{b_r} to both Monitor and Video, and in having a limited window area.

Monitor transformations

Information is communicated to the monitor during each of the S-M-V operations. The area of interest in an image is specified by a 'window' at the S-M-V control grid delineated by the Begin and End Coordinates. Three types of transformations may be applied to the window as viewed at the monitor: rotation, magnification and translation.

Rotation

The Rotation parameter, G, allows the option of:

(R1) slaving the monitor to the S-M-V Controller grid in scan axis and direction or,

(R2) choosing the scan axis and direction at the Monitor to be parallel to the X-axis and incrementing irrespective of the Controller choices.

When option (R2) is taken then scanning the X-axis at the Controller grid obtains the transformations shown in Figure 10, whereas scanning the Y-axis yields the transformations shown in Figure 11. The choice of a B-E orientation fixes the scan direction and the initial scan line, hence selects one of the four transformations. The four transformations in Figure 10 are called the "four group" of rotational symmetries on the rectangle. The eight transformations in Figures 10 and 11 define the "Klein Rotation Group" of symmetries on the square.

Magnification

The window displayed at the monitor can be magnified by factors of 2 with the parameter *h* subject to the combined restrictions:

$$h + \begin{Bmatrix} p \\ q \end{Bmatrix} \leq \begin{Bmatrix} p_{max} \\ q_{max} \end{Bmatrix}$$

The underlying assumption is that the range of resolution options on p and q is identical at monitor and scanner, and that the unit cell at monitor is magnified by $m = 2^h$.

One naturally constrains the choice of *h* to keep the magnified window from exceeding the raster area:

$$m \cdot \begin{Bmatrix} |XB\text{-}XE| \\ |YB\text{-}YE| \end{Bmatrix} \leq 1.$$

This restriction is refined in the discussion below on translation.

Translation

Translation of the window at the monitor may be achieved with the *Monitor Displacement Coordinates.*

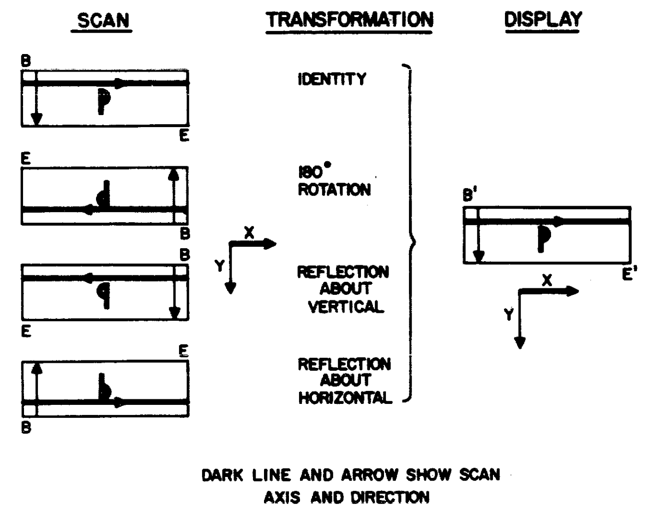

SCAN
TRANSFORMATION
DISPLAY
IDENTITY
180° ROTATION
REFLECTION ABOUT VERTICAL
REFLECTION ABOUT HORIZONTAL
B
E
B'
E'
X
Y
DARK LINE AND ARROW SHOW SCAN AXIS AND DIRECTION

Figure 10—Group transformations effected by the four different Begin-End coordinate orientations with X-axis scan. (G = 1)

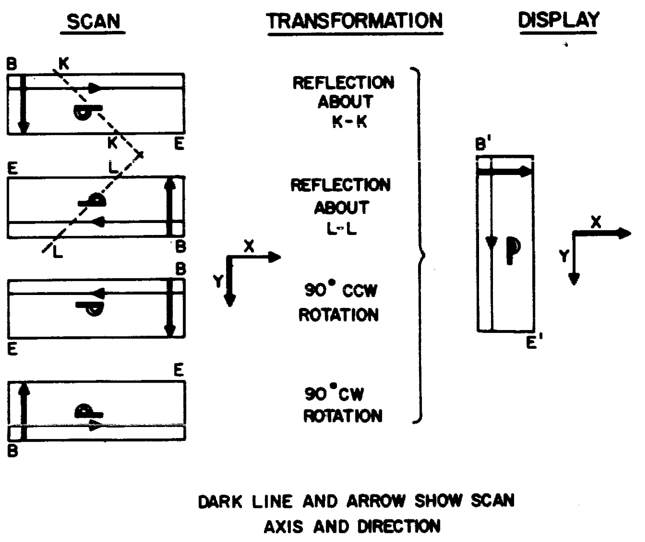


Figure 11—Same as Figure 10 with Y-axis scan. (G = 1)

As shown in Figure 12, D transforms into (0, 0) at the monitor, hence B is repositioned accordingly.

If a magnification m is superimposed on the translation it affects the area delineated by the D-E coordinates, hence the combined transformation is completely defined as operating on the two vectors **u** and **v**:

1. translate the tail of **u** to (0,0) and
2. magnify the length of **u** and of **v** by m.

The four allowed orientations of D, B and E are shown in Figure 13. The implied constraint is that in case (a) D $\leq$ B $\leq$E with a corresponding interpretation for the other three cases. D always transforms into the

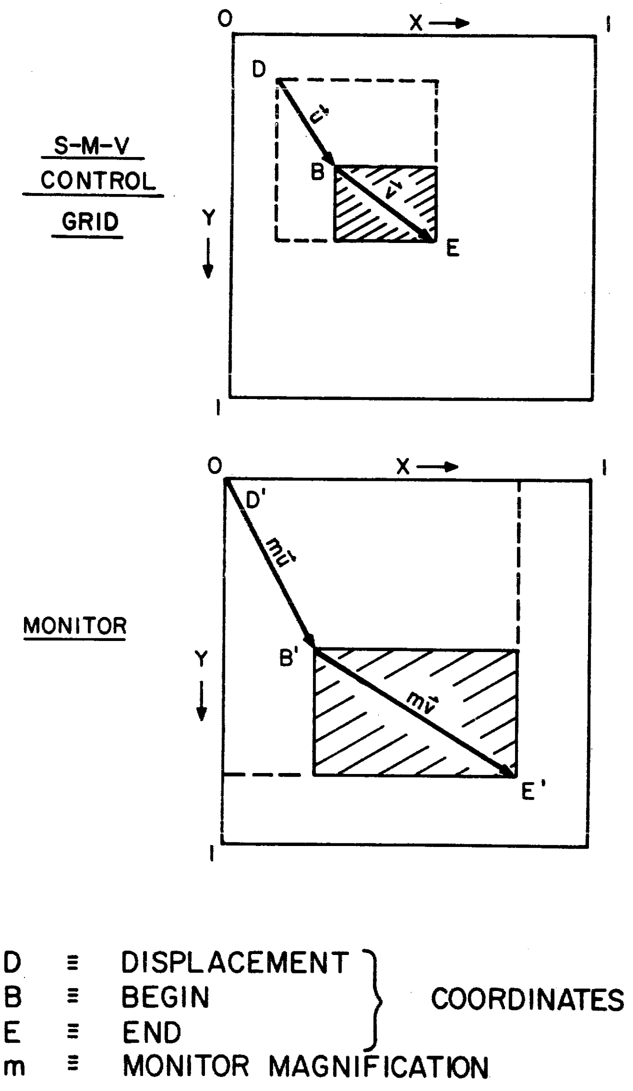


Figure 12—Image translation and magnification at the monitor

corresponding corner position at the monitor with Rotation option (R1). For Rotation option (R2), D goes to (0, 0) at the monitor in all cases.

For proper centering, one must choose (XD, YD) such that

$$m\,(\,|XB - XE| + 2\,|XD - XB|\,) = 1$$

and

$$m\,(\,|YB - YE| + 2\,|YD - YB|\,) = 1.$$

Complete positioning freedom is not always possible when combined with one of the rotation transformations, e.g., when the window is very close to the edge of the grid and the chosen transformation requires D to be on the edge side of the window.

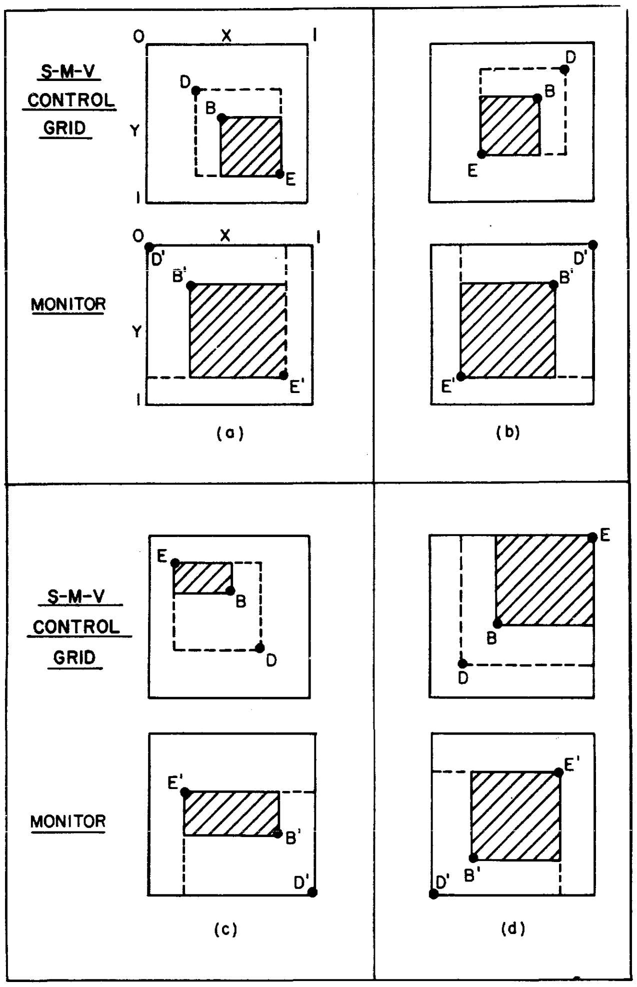


Figure 13—Four allowed orientations of D, B and E coordinates with the corresponding monitor interpretation. (G = 0)

Totally slaved to scanner

When tracking a line or a boundary, by scanning a sequence of windows, a 1-1 correspondence between Monitor and source is desirable; otherwise the relative positioning of windows at the source is not reflected at the Monitor, and the tracking procedure cannot be viewed. This can of course be achieved with the proper parameter assignments as a standard transformation but one would like to avoid the time involved in doing so. Since the window will be small the scanning time can be significantly reduced by recognizing a special case. The following natural setting for the parameters listed can be interpreted as defining the special case:

1. (XD, YD) = (0, 0), no translation
2. gross resolution
3. no rotation, option (R1)
4. no magnification

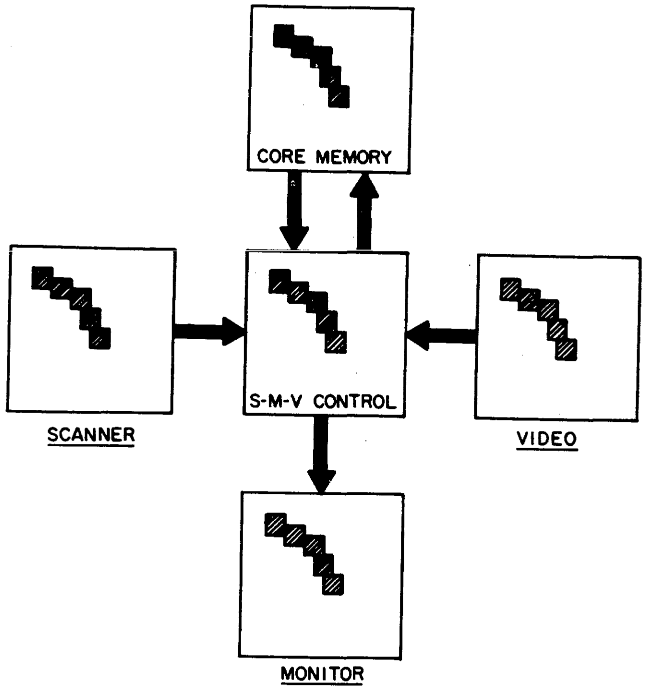


Figure 14—Monitor totally slaved to the source media

As shown in Figure 14, the Controller can then avoid the time-consuming redundancy of the transformation steps inherent in a sequence of small windows.

Totally slaved to video

As indicated in Figure 14, video is included in the total slaving concept. For video this is accomplished by replacing constraint (4) in the previous section with the following:

$$(4') \quad h \cdot \Delta X = \Delta X_{max} (j,n).$$

At most one window can be scanned per video frame, thereby limiting the repetition rate.

Video transformations

The video network, unlike the monitors, can act both as a source and as a destination. A window, specified by the Begin and End coordinates at the S-M-V control grid, determines the area of interest. Transformations similar to those at the monitor can be effected within the limits of the video constraints.

Source medium options

It is useful to distinguish two interpretations of video as a source medium:

(S1) 1-1correspondence between video and the S-M-V control grid (excluding rotation), thereby allowing translation and magnification of a window to the monitor;

(S2) 1-1 correspondence between video and some portion of the S-M-V control grid, effectively allowing translation and *de*magnification of a window to film.

Options (S1) and (S2) are illustrated in Figures 15 and 16 respectively.

Rotation

When video acts as a destination medium, the transformations are effected in the same manner as they are to the monitor under option (R2). Because the video scan axis and direction are fixed, option (R1) never applies.

When video participates as a source medium the role is reversed and one always gets the inverse transformation to the S-M-V control grid. If option (R2) is chosen for the monitor the two transformations cancel (into and then out from the control grid)—effectively yielding the identity transformation to the monitor.

Magnification and demagnification

Equation (6) defines $(\Delta X)_{max}$ (j,n), the largest useable ΔX. Choosing $\Delta X < (\Delta X)_{max}$ (j,n) allows a video magnification of

$$m_v(j,n) = \frac{(\Delta X)_{max}(j,n)}{\Delta X} = 2h_v(j,n)\ ,$$

so

$$h_v(j,n) = b_g - j + n - p = p_{max}(j,n) - p\ .$$

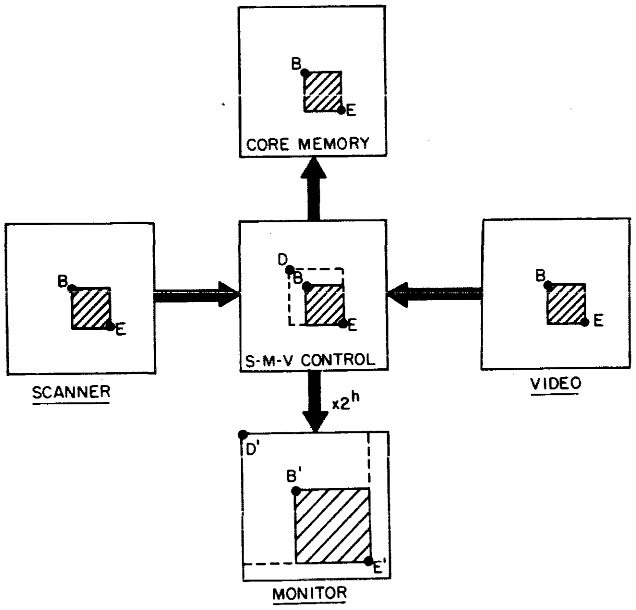


Figure 15—Magnifying a window at the monitor and/or transmitting to core memory

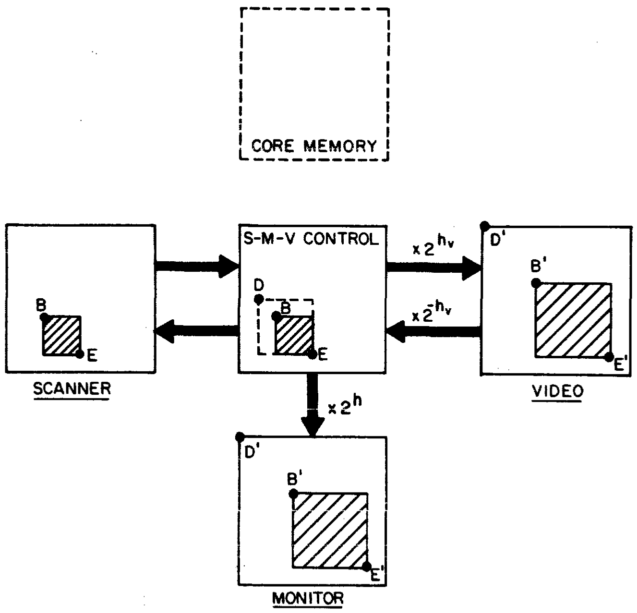


Figure 16—Transmitting between scanner and video. Display at the monitor with h = h_v

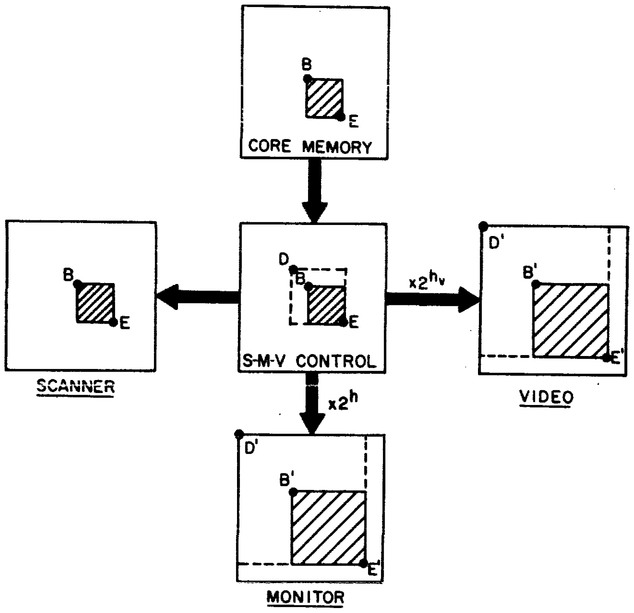


Figure 17—Transmitting from core memory to one or more media. If to video, h = h_v

When video is a source medium m_v is a demagnification, and when video is a destination medium m_v is a magnification.

In the (S1) interpretation of video as a source medium $\Delta X = (\Delta X)_{max}$ is assumed, and the monitor magnification is achieved in the same manner as when the source is core memory or film scanner.

Translation

When video is a destination medium translation is effected in the same way as it is at the monitor with the rotation option (R2); (XD, YD) at the S-M-V control grid transforms into (0, 0) at the video. For proper centering (XD, YD) is chosen so that:

$$m_v(|XB\text{-}XE| + 2|XD\text{-}XB|) = S(j,n) \cdot \Delta X \leq 1,$$

$$m_v(|YB\text{-}YE| + 2|YD\text{-}YB|) = \frac{\Delta Y \cdot N_u(i)}{(\Delta Y)_v} \leq 1,$$

where $N_u(i)$ is the number of useable lines per video frame (or maximum $(\Delta Y)_v$ steps).

With video as a source medium translation to the monitor is accomplished by associating (0, 0) at the video with (0, 0) at the S-M-V control grid. Translation to the monitor then takes place in the usual way. This is option (S1) and is illustrated in Figure 15. Option (S2) is accomplished by associating (0, 0) at the video with (XD, YD) at the S-M-V control grid and is illustrated in Figure 16. Translation to film then takes place as the inverse of the translation effected when the transfer is from film to video. Note that translation to monitor and film cannot be effected simultaneously; the options (S1) and (S2) are mutually exclusive as far as translation is concerned—hence the distinction.

Totally slaved as a destination

Video is not likely to be useful for display under the totally slaved concept, since this would constrain the sampling increment to be

$$\Delta X = \Delta X_{max}(j,n).$$

SUMMARY

This paper has developed the parametric description of a general purpose Scan/Display System for image digitization and display. Central to the system is the S-M-V Controller which can service either simultaneously or individually three distinct media: film, closed-circuit television and incrementally-driven CRT displays. An adjunct of the system is a Video Communications Net to provide both high and commercial resolution service to remote users.

Media compatibility

The S-M-V Controller acts as a media-media interface that identifies the necessary information transfer constraints or rejects the operation request as one demanding inconsistent parameter assignments. Transfer constraints considered include X/Y resolution ratios, aspect ratios and line sweep times of the media, and D-A/A-D conversion times. A constant data rate option allows operation at I/O channel capacity for all choices of gray scale resolution.

The digital encoding of an image generated in a scanning operation can be retransmitted to the S-M-V Controller for output (display)—and on any of the three available media.

Rasters

By associating (X,Y) positions with binary counter value pairs the controller can generate a family of the two-dimensional regular lattices—rectangular and hexagonal. X and Y resolutions are independently variable, the allowed resolution values are in geometric progression and correspond to changes in counter incrementing position. A selected "window" of the full image can be specified.

Sampling/display strategies

Three sampling formats (Raster, Coordinate and Incremental) allow a variety of sampling/display strategies. Raster format provides uniform sampling of all lattice positions. For coordinate format however, sampling takes place only at those lattice positions where the image satisfies some criteria prescribed by selection of a triggering/filtering circuit.

Incremental format is provided for segment vector plotting. Commands are provided for setting the starting point, line width and plotting resolution. Segment iteration can be specified.

The sampling beam stencil is variable in size, shape and orientation. The shape options are spot/slit. The slit option includes orientation resolution and range.

Metrological facilities

An optional local extension of position resolution can be specified through a gross/vernier counter selection. With the vernier option selected, the gross counters define a benchmark while the vernier counters represent a local displacement. Using this technique, positional resolution of 1: 30,000 is currently attainable in flying spot scanning.

Implementation

The Illiac III computer[9,10] employs a scan/display system with parameters as specified in Appendix C of this paper.

Except for the video scan converter, the microimage storage and the video storage, the scan/display system is anticipated to be operational by Summer 1969.

ACKNOWLEDGMENT

Many stimulating discussions with members of the Illiac III staff aided in the formulation of concepts developed in this paper. Mr. Robert C. Amendola has contributed significantly to the Video Network specifications and to scanner optical design. Dr. Kenneth J. Breeding participated in the first design of a scanner controller which was subsequently expanded into the S-M-V controller described in this paper.

A description of the analog and digital logic design of the scan/display system is now being prepared for publication by Dr. James L. Divilbiss and Mr. Ronald G. Martin, respectively.

The authors wish to thank Mr. John H. Otten for preparing the illustrations and Mrs. Donna J. Stutz for typing the paper.

REFERENCES

1 L A DUNN L N GOYAL B H McCORMICK V G TARESKI
S-M-V programming manual
Department of Computer Science University of Illinois Urbana Illinois March 1968

2 D M COSTIGAN
Resolution considerations affecting the electrical transmission of technical documents by scanning process
National Microfilm Association Jour Vol 1 No 3 Spring 1968

3 B F WADSWORTH
PEPR—a hardware description
Emerging Concepts in Computer Graphics Don Secrest and Jurg Nievergelt (Eds) W A Benjamin Inc New York 1968

4 R CLARK W F MILLER
Computer-based data analysis systems
Methods in Computational Physics Vol 5 Berni Adler Sidney Fernbach Manuel Rotenberg (Eds)
Academic Press New York 1966

5 R B MARR G RABINOWITZ
A software approach to the automatic scanning of digitized bubble chamber photographs
Methods in Computational Physics Vol 5 Berni Adler Sidney Fernbach Manuel Rotenberg (Eds)
Academic Press New York 1966

6 J VANDER LANS J L PELLEGRIN H J SLETTENHAAR
The hummingbird film digitizer system
SLAC Report No 82 Stanford Linear Accelerator Center Stanford University Stanford California March 1968

7 R S LEDLEY L S ROTOLO T J GOLAB J D JACOBSEN M D GINSBERG J B WILSON
FIDAC: Film input to digital automatic computer and associated syntax-directed pattern recognition programming system
Optical and Electro-optical Information Processing Teppep J Berkowitz D Clapp L Koester C and Vanderburgh Jr (Eds) M I T Press Cambridge Massachusetts 1965 Chap 33

8 H KAZMIERCZACK F HOLDERMANN
The karlsruhe system for automatic photointerpretation
Pictorial Pattern Recognition G C Cheng R S Ledley Donald K Pollock and A Rosenfeld (Eds)
Thompson Book Co Washington D C 1968

9 B H McCORMICK
Advances in the development of image processing hardware
Image Processing in Biological Science Ramsey D M (Ed) University of California Press 1968 in press

10 B H McCORMICK
The illinois pattern recognition computer—illiac III
IEEE Transactions on Electronic Computers Vol EC-12 No 5 December 1963

APPENDIX A—DEVICE SPECIFICATIONS

Scanners

The scanners are flying spot scanning systems with an added diquadrupole coil for astigmatic defocusing of the spot into a line element to achieve a slit mode. All scanners are capable of either scanning from developed film or photographing onto unexposed film. The optical path of the beam is split, with one path transversing the film and the other path through a reference grid to establish stability against engraved fiducial marks.

Several types of media transports are provided to handle the projected range of problems. A 70 mm. scanner is provided primarily for 70 mm. negative bubble chamber film. Here the format of the raster is 2.362 inches × 3.522 inches, and the minimum spot size is approximately 0.001 inch at the film. Due to the length of the frame to be scanned, scanning is done in two steps. The two horizontal halves of the frame are scanned successively with a 4 mm. overlap to establish half-frame continuity. Large motors are used for slew and gross positioning of the film and a small digital stepper motor is used for fine positioning of the frame. Frame position sensing is accomplished by using the digital stepper motor as a tachometer and by counting sprocket holes. Total film capacity is 1000 feet.

A scanner for handling 47 mm. film is similar to the 70 mm. transport design except for the following: The film format is different. A friction drive is used on the digital stepper motor, since the film is unsprocketed. The frame position is determined by sensing small index blocks at the lower edge of the film using a fibre-optics light guide and a photodiode.

The microform scanner contains three units. The first is a 35 mm. full frame digitally controlled camera which can read light through the film both negative and positive. The second unit contains a 16 mm. Bolex camera for making computer-generated black and white movies and a modified 16 mm. film editor for scanning 16 mm. film of all types. The third unit is a microfiche transport mechanism for scanning and producing a single microfiche in the 72 image COSATI format. For the three different units the C.R.T. raster is adjusted optically to fit the particular frame size.

A fourth type of scanner is built around a microscope with a digitally controlled automatic stage. Positional accuracies are on the order of $\pm$ 2 microns, and the maximum slide area coverage is 1.2 inches $\times$ 1.2 inches. Variable reduction is available from a four objective rotating turret. Full visual observation is available to an operator.

Monitors

The monitors consist of 21-inch cathode ray tubes controlled in a manner similar to the scanner C.R.T.'s; viz., digital position counters control the spot location through accurate, high-speed digital-to-analog converters. The monitor counters are digitally controlled directly from the S-M-V Controller via an incremental communications scheme; essentially the only commands issued by the S-M-V Controller for the monitors are increment the counters, decrement the counters, reset the counters, and reset the parameters. Therefore, any spot movement possible on a scanner C.R.T. can be accomplished on the monitor C.R.T. The video input for the C.R.T. grid is also synchronized by the S-M-V controller.

Included with the monitors for communications to a central processing system are a selectric typewriter, microtape input/output tape drives, and a light pen for cursor control.

Video scan converter

The video scan converter consists of a high resolution storage tube capable of storing a useable picture for at least 30 seconds. Multiple readouts can be made from a stored image before degradation is significant.

The storage tube can be written into and read from at any of the video rates in the system (525, 1536F, 1536S) on the video switching matrix side. On the SMV control side the scan converter looks like a film as seen by a scanner; therefore, reading and writing is handled in exactly the same manner as it is in a scanner.

Video switching matrix

The video switching matrix is a mechanical cross bar matrix. Therefore, the switching speed will be in the order of 100 milliseconds or less. In this routing switch any source can be switched to from one to three different destinations simultaneously. In addition, switching provisions are also included to mix any two video sources to provide a composite signal to the selected destinations.

Character generator

The Character Generator is designed to accept up to 512 ASCII characters into its 4096 bit memory. A 99 dot matrix, 9 dots wide by 11 dots high, is used to develop each character into the appropriate video levels. The maximum TV screen display is 16 horizontal rows of 32 characters or spaces each. Alternatively 132 characters/line print-out can be generated on the Videograph printer. A special cursor is also available along with eight commands for controlling it.

The output composite video signal can be either 525 or 1536 lines per frame, depending upon the externally supplied sync signal.

Videograph printer

The Videograph Printer can print on demand at a rate of 0.8 seconds per 8½ $\times$ 11 inch sheet. Horizontal resolution is 128 lines per inch and vertical resolution matches the high resolution of the 1536 line slow CCTV cameras. Gray scale resolution is limited to four shades. The paper used is inexpensive zinc-oxide coated stock.

525 line T.V. cameras and monitors

The 525 T.V. Cameras and Monitors are conventional television units; namely, 525 lines per frame, 30 frames per second interlaced (60 fields per second). These units provide for relatively low cost reduced resolution, which is sufficient for many message routing and simple acquisition and display purposes.

1536 F/S cameras

The 1536 F/S cameras are vidicon camera units which can be remotely selected to operate either in fast scan mode (15 frames per second) or slow scan mode (1.25 frames per second). The format of either mode is 1536 lines per frame done in a sequential (non-interlace) scan. The aspect ratio is variable, but it is set for a nominal $8\frac{1}{2} \times 11$ aspect ratio. The camera bandwidth is limited to 9.5 Mhz for fast scan and 1.4 Mhz for slow scan.

Remote video consoles

Each remote console is a self-contained unit with two video monitors and the necessary equipment for communicating with a digital computer. The video monitors consist of a 17 inch 1536 lines per frame slow (1.25 frames per second) monitor with a P-26 phosphor and a 17 inch 1536 lines per frame fast (15 frames per second) monitor with a VC-4 phosphor. Each monitor matches characteristics of the associated camera.

Included with the console for direct digital communications to the central computer are a teletype ASR-33 unit and a small special keyboard to be used for frequently used machine orders. Other items to be included with the consoles are a microfiche reader, a digital patch panel for digital control signals, and an analog patch panel for analog control signals.

Special plug-in options for a universal cursor control could provide for such devices as a light pen, joy stick, matrix pad, bug, etc. Other options could include provisions for direct handwriting of orders at the console by T.V. camera pick-up and/or Rand tablet type device and a monitor microfiche camera for filming images from the C.R.T. screen.

Microimage storage

The Microimage storage consists of a microfiche reader/access mechanism that is able to store, retrieve, and display COSATI standard microfiche on demand. Storage of the microfiche is by a rotary drum that is a changeable unit. Images can be digitally selected, and the display of any requested image requires less than five seconds. Access time to an adjacent image (i.e., one within the same fiche) is less than two seconds. The output is displayed in an $8\frac{1}{2} \times 11$ inch format if desired, and it is projected upon the two-inch vidicon of a 1536 F/S line television camera for distribution into the video network.

The drum holds 750 modified microfiche cards of 60 frames each. Each frame again contains an array of 72 microimages or basically a standard microfiche. Therefore, a single drum will provide storage for 3,240,000 page images. Readout from the selected microfiche frame is accomplished with a fly's eye readout mechanism.

Video storage

The video storage consists of an interchangeable 72 track video disk, where a single track can contain a complete image. Input and output can be at either the 1.25 or the 15 frames per second rate with resolution matched to the corresponding video devices.

APPENXIX B—SYMBOL AND PARAMETER LIST

$	Parameter values assigned at design time
*	Parameter values explicitly assigned at execution time

Upper case Latin

*	A	Scan *A*xis selection
	ACW	*A*ngle *C*oordinate *W*ord
	B	Equivalent to (XB, YB)
*	B_s	Standard spot/nonstandard spot, or slit selection
	BCW	*B*egin *C*oordinate *W*ord
*	C	Media selection, *C*ore memory
	D	Equivalent to (XD, YD)
	DCW	*D*isplay *C*oordinate *W*ord
*	DF	*D*ata *F*ormat selection
	DPB	*D*isplay *P*arameters *B*yte
	DX	*X*-component of the incremental format *D*isplacement vector
	DY	*Y*-component of the incremental format *D*isplacement vector
	E	Equivalent to (XE, YE)
	E_v	Same as E, to distinguish *V*ernier
	ECW	*E*nd *C*oordinate *W*ord
*	F	Media selection, *F*ilm (scanner)
	FPB	*F*ormat *P*arameters *B*yte
*	G	*G*roup rotation selection for the monitor
*	J	Display regeneration request
*	K	Constant data/sample rate option (for a given p) with the raster format
*	L	*L*attice selection
	L_s	*L*ength of *S*lit = 2^v gross basic cells
	LPB	*L*attice *P*arameters *B*yte
$	N(i)	*N*umber of lines per video frame
	N_u(i)	Number of useable lines per video frame (or maximum $(\Delta Y)_v$ steps).
	N_b	*N*umber of *B*its in a raster string representation
	N_s	*N*umber of *S*amples in a raster string representation
	NGL	*N*umber of *G*ray Scale *L*evels = 2^{2^n}
	PW	*P*arameter *W*ord
*	R	Gross/vernier *R*esolution selection
*	S	*S*equence selection, sequential/interlaced

	S(j,n)	Maximum achievable number of *S*amples per full video scan line
	$S_{max}(j)$	Maximum value of S(j,n) (achieved for n = 0)
	SPB	*S*lit/spot *P*arameters *B*yte
	SX	*S*ign of D*X*
	SY	*S*ign of D*Y*
*	T	*T*rigger (filter) selection for encoding in the coordinate string representation
*	V	Media selection, *V*ideo
	V_s	*S*weep *V*elocity
	W_s	*W*idth of *S*lit or spot diameter = 4^u gross basic cells
*	XB	*X*-coordinate, *B*egin cell
*	XD	*X*-coordinate, *D*isplacement
*	XE	*X*-coordinate, *E*nd cell
	XE_v	Same as XE, to distinguish *V*ernier
*	YB	*Y*-coordinate, *B*egin cell
*	YD	*Y*-coordinate, *D*isplacement
*	YE	*Y*-coordinate, *E*nd cell
	YE_v	Same as YE, to distinguish *V*ernier

Lower case Latin

$	b_a	Number of bits in the angle orientation counter
$	b_c	Number of interpolation bits concatenated with the b_g-p gross resolution bits
$	b_g	Number of bits in the gross position counter
$	b_o	Number of gross-vernier overlap bits
$	b_r	Number of vernier bits that extend gross resolution
$	b_v	Number of bits in the vernier position counter
	c_1	Sweep velocity constant (with $c_1 = f_c$, V_s is given in basic cells per microsecond)
$	f_b	Maximum data bit transfer rate
$	f_c	Position counter incrementing frequency
	f(j)	Video unit cell match function = $r_v/r_s \cdot N(i) \cdot 2^{-j}$
	g(j)	Video line selection modifier. Closest interger such that $f(j) \simeq 2^{g(j)}$
*	h	Monitor magnification = 2^h
$	h_{max}	Maximum value of h
	$h_{v(j,n)}$	Video magnification/demagnification exponent (see $m_v(j,n)$)
	i	Video system identification
	j	Video system resolution parameter, $S_{max(j)} = 2^j$
	k	k(R): k = 0 for gross; b_r for vernier
	m	Monitor magnification = 2^h
	$m_{v(j,n)}$	Video magnification/demagnification = $2^{h_v(j,n)}$
*	n	2^n is the number of bits of gray scale
$	n_{max}	Maximum value of n
*	p	$\Delta X = 2^p$ (see ΔX)
	$p_{max}(j,n)$	Maximum value of p for Video Network $= b_g - j + n$
*	q	$\Delta Y = 2^q$ (see ΔY)
$	r_s	Aspect ratio, control grid (*S*tandard)
$	r_v	Aspect ratio, video
*	s_v	Sign bit of vernier counter
*	u	Slit width is 4^u gross basic cells
$	u_{max}	Maximum value of u
*	v	Slit length is 2^v gross basic cells
$	v_{max}	Maximum value of v
*	z	$\Theta = 2^{z-b_a}$ (see $\Delta\Theta$)
$	z_{max}	Maximum value of z

Greek

	ΔC	Counter increment along the scan axis in basic cells
	$\Delta t(i)$	Time to sweep one video scan line
	ΔX	Sampling increment along the X-axis at the S-M-V control grid ($\Delta X = 2^p$)
	$(\Delta X)_{max}(j,n)$	Maximum value of ΔX corresponding to S(j,n)
	ΔY	Sampling increment along the Y-axis at the S-M-V control grid ($\Delta Y = 2^q$)
	$(\Delta Y)_v$	Sampling increment along the Y-axis at video (every $(\Delta Y)_v$ lines)
	$\Delta\Theta$	Orientation sampling ($\Delta\Theta = 2^{z-b_a}$ units of angle)
	$\delta\mathbf{x}$	X-axis unit vector
	$\delta\mathbf{y}$	Y-axis unit vector
*	ΘB	Orientation Begin value
*	ΘE	Orientation End value

APPENDIX C—DESIGN VALUES FOR THE ILLIAC III SCAN-DISPLAY SYSTEM

b_a	angle orientation counter	8 bits
b_c	interpolation bits concatenated to gross	3 bits
b_g	gross position counter	12 bits
b_o	gross-vernier overlap	4 bits
b_r	vernier resolution extension to gross	3 bits
b_v	vernier position counter	7 bits
f_b	maximum data transfer rate	10 Mhz
f_c	counter incrementing frequency	1.25 Mhz
h_{max}	2^h is image magnification at monitor	7

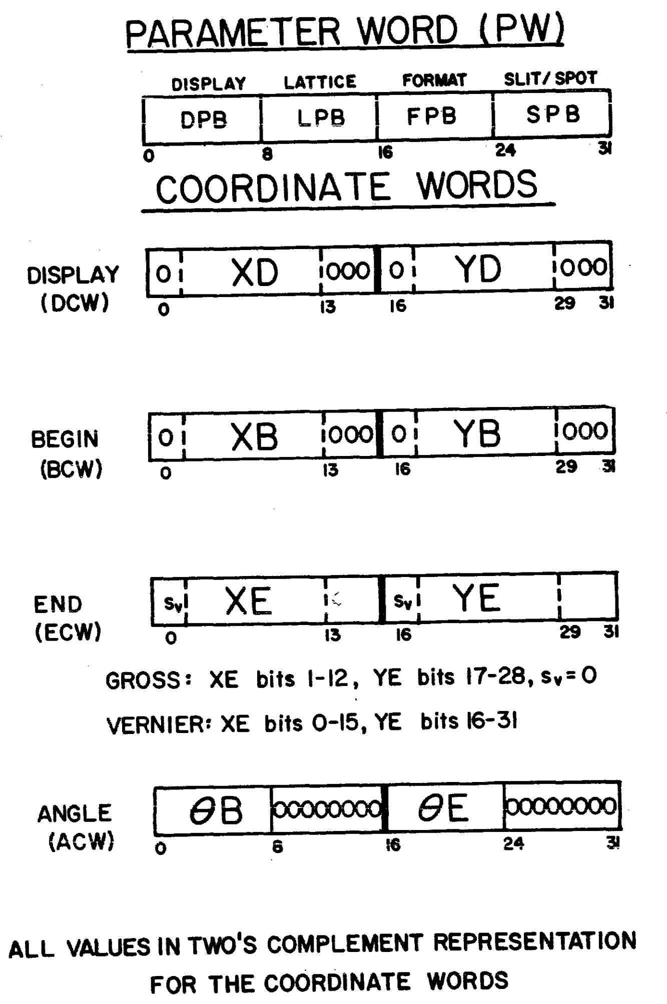


Figure 18—Parameter and coordinate words formats as defined for Illiac III

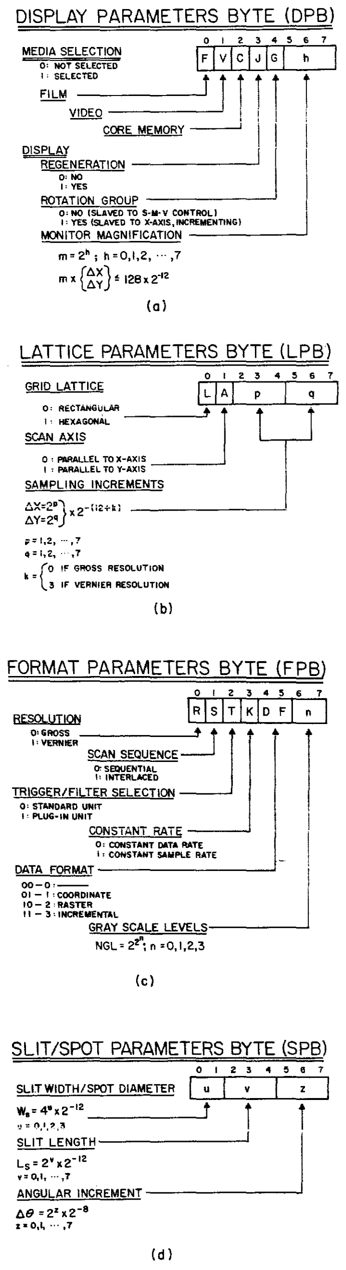


Figure 19—Display (a), Lattice (b), Format (c), and Slit/Spot (d) Parameter Bytes as defined for Illiac III

n_{max}	$2^{n_{max}}$ is maximum bits of gray scale	3
p_{max}	X-axis sample increment is 2^p basic cells	7
q_{max}	Y-axis sample increment is 2^q basic cells	7
r_s	Aspect ratio, control grid	1:1 (but variable)
r_v	Aspect ratio, Video	3:4

u_{max}	slit width is 4^u basic cells	3
v_{max}	slit length is 2^v basic cells	7
z_{max}	angle increment is 2^{z-b_a} units	7

Interactive tolerance analysis with graphic display

by LAWRENCE A. O'NEILL

Bell Telephone Laboratories, Incorporated
Holmdel, New Jersey

INTRODUCTION

One of the principal advantages of a dedicated computer with on-line graphic display is that the user is able to quickly and easily interact with the program. The value of this interaction has been clearly demonstrated in the optimization of circuit and system designs.[1,2] A related task on which interaction can have a significant impact is the investigation of the influence of parameter tolerances on the performance of an optimized design.

There are three basic approaches to computer-aided tolerance analysis currently in use—moment method, monte carlo approach, and worst case analysis.[3,4] The experimental approach presented here is a version of the monte carlo approach in which performance distributions are acquired by randomly perturbing the parameters. It is economically feasible to accumulate distributions rather than just compute a few statistics because of interaction and graphic display. The experimental approach is not subject to the linear assumptions of the moment method and the performance estimates are more realistic than those obtained by worst case. The advantages of the experimental approach, in computational simplicity and in the nature of the design information provided, are discussed and illustrated in this paper.

Tolerance analysis

Tolerance analysis[3] of systems and circuits is required because the optimized parameter values cannot be exactly realized in manufacture, and the range of conditions under which the system must operate is usually much broader than the limited number of cases that can be considered during optimization. The parameter values are not exact because components are usually available only in discrete ranges with a statistical spread in each range that is dependent upon the manufacturing process and the component's age. Operating conditions are usually introduced during optimization as either typical or worst case discrete values, but more realistic estimates of anticipated performance can be obtained if the actual statistical distributions are taken into account.

Tolerance analysis provides two primary sources of information to the designer:

1. Tolerance Specifications—The designer can specify maximum tolerance limits guided by a prediction of the expected yield.
2. Design Information—The designer can select among alternative realizations based either on which is less sensitive to expected parameter variations, or which will be less expensive to manufacture.

Experimental approach

The experimental approach to tolerance analysis consists of simulating the system under investigation, randomly perturbing the parameter values and displaying the distribution of performance criterion. The two basic advantages of this approach are:

1. Nonlinear systems can be investigated as easily as linear ones, without making assumptions and approximations.
2. The distribution of the performance criterion that is necessary for further statistical investigations can be measured.

This nonlinear capability is important not only because many systems contain nonlinear elements but also because nonlinear performance criterion are normally applied even to linear problems—for example, a mean squared error criterion. A nonlinear criterion complicates a statistical investigation because the relative influence of parameter tolerances cannot readily be predicted, but the influence can be measured by the experimental approach. A measured frequency distribution, when properly normalized, provides the probability used in statistical estimates of yield and cost.

Without interaction, the computer run times required using this approach become prohibitive because a large number of trials is needed to acquire confidence that an histogram is representative of the population. In addition, the time required to evaluate the performance is frequently long. With interaction and graphic display, the number of trials can be significantly reduced by quickly terminating unsatisfactory distributions and also recognizing when sufficient samples have been accumulated by the insensitivity of the display to new data. Moreover, if the dedicated computer can be operated in a hybrid mode, the solution time for problems requiring time domain analysis can be drastically reduced by employing analog simulations. The ability to use whichever mode of simulation (digital or hybrid) best satisfies the requirements greatly broadens the range of problems to which the experimental technique can be economically applied.

Tolerance analysis program

The experimental approach has been implemented with a digital program. The basic features of this program are shown in Figure 1.*

The parameters of the given system are randomly *perturbed* according to known statistical distributions and the *performance* is measured. The perturbation subroutine provides for the shaping of the distribution obtained from the random number source and for the correlation of the independent numbers to satisfy specific problem requirements. The distribution of the performance criterion is obtained from these measurements by dividing the criterion range into class intervals and accumulating the number of occurrences in each interval. The accumulated data may be *displayed* either as a frequency distribution histogram or a cumulative distribution histogram. The real-time display on a digital scope enhances the evaluation of the histogram and facilitates *interaction* with the program. Documentation is provided on the printer—both parameter specifications and the distributions are printed when required. The real-time interaction allows the user to modify the histogram scaling, parameter nominal values

* A detailed description of this program has been submitted, elsewhere, for publication by D. M. Bohling and the author. In that paper, the procedures for random number generation, histogram accumulation, and interaction with the computer are discussed.

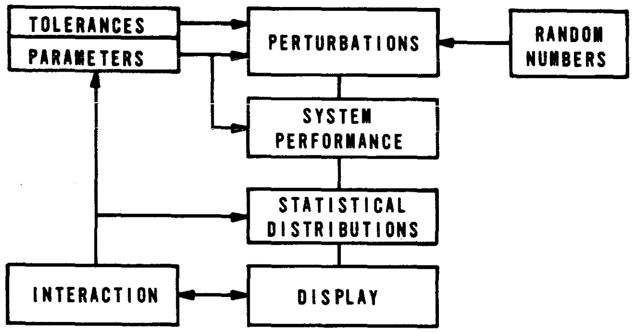


Figure 1—Tolerance analysis program

and tolerances whenever required so that computer time is not wasted in accumulating useless data. It is the simplicity with which this technique can be applied to any problem, linear or nonlinear, without extensive, preliminary investigations that led to the design and wide utilization of this tolerance analysis program.

Application

The discussion of the experimental approach to tolerance analysis can be clarified by considering applications that illustrate its features. The two examples were selected to demonstrate the value of being able to use both digital and hybrid simulations, and the value of the ability to measure performance in nonlinear problems and then accumulate statistical distributions.

Digital simulation of an inductor

The first example illustrates the application of the tolerance analysis program to an all-digital simulation of a gyrator circuit. The objective of this experiment was to determine what component tolerances would be necessary in the gyrator circuit of Figure 2 to allow it to be used as a replacement for an inductor. The reason for using this circuit is to save on the cost, weight, and size of the inductors that are used in this frequency range. Such a replacement is feasible because the price of the integrated circuit components and amplifier to be used are expected to become quite inexpensive in the future.

The gyrator circuit must exhibit the same characteristics as an inductor and be stable, throughout the frequency range from .5 to 10 KHZ. The performance criterion selected was the Q of the circuit—the ratio of standard measure of inductor quality. The Q was evaluated by calculating the input impedance of the gyrator and taking the ratio of imaginary to real part of this impedance. The circuit will become unstable if the real part of the impedance becomes negative. At the instability threshold, the Q becomes quite large and thus

1/Q was used as the criterion to simplify scaling. Since the evaluation of this impedance at a particular frequency is an algebraic problem, one can take advantage of the high speed of digital simulation. A subroutine was written that calculated this performance criterion directly from the perturbed component values.

The display of the frequency and cumulative distributions of 1/Q used in evaluating the performance is shown in Figure 2. Gaussian variations were applied to all parameters on the circuit—passive components and amplifier characteristics. Several sets of component tolerances were evaluated before the acceptable performance illustrated in Figure 2 was obtained. The frequency distribution was used to determine the stability of the realization (no designs should exist with negative values of 1/Q) and to acquire confidence in the sample size. The cumulative distribution reveals what percentage of the designs can be expected to have a Q less than any selected value. The tolerances used for this design were: 0.1 percent on the passive components, and for the amplifier characteristics, 50 percent on the gain, 10 percent on the cutoff frequency, 67 percent on the input resistance and 100 percent on the output resistance. The design was then statistically evaluated at several frequencies within the intended operating range.

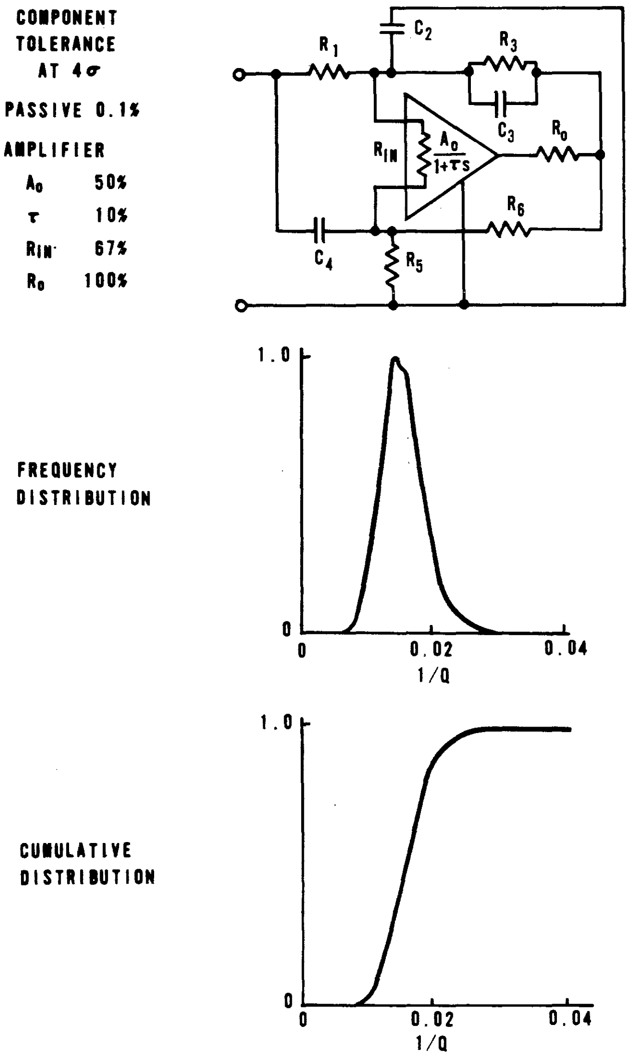


Figure 2—Performance of a gyrator simulation of an inductor may be evaluated from histograms obtained by applying Gaussian distributions to components

The accumulation of data with this digital simulation was extremely fast; 200 designs could be tested and the histogram updated every second. The total amount of computer time required for this investigation was quite short not only because of this fast operation, but also because the interactive capability allowed the user to terminate unsatisfactory designs before many samples were accumulated.

Hybrid simulation of a pulse equalizer

The influence of parameter tolerance on the error rate to be expected from a pulse equalizer was investigated to establish manufacturing specifications. The equalizer was one of a set to be used in a digital transmission system operating at 6.3 megabits per second over standard telephone lines. The nominal equalizer designs were optimized by Fleischer[2] on the same interactive, hybrid facility that was used for tolerance analysis; therefore, the basic analog simulation and digital error rate calculations were available.

The calculation of error rate includes terms reflecting intersymbol interference, thermal noise, crosstalk and sampling jitter. The transient response of each equalizer design had to be evaluated to determine interference and noise. The intersymbol interference, which indicates the effect of adjacent pulses on the one being equalized, was calculated from the analog measurements of the signal at sample points preceding and following the pulse peak. The thermal noise power to be expected from each design was estimated from the impulse response of the system. The determination of the transient response by digital computer simulation took far too long for either optimization or tolerance analysis and thus an analog computer simulation was used.

The block diagram of the hybrid simulation of the equalizer that was used in the tolerance analysis is shown in Figure 3. The computer program repeats the following sequence of operations and accumulates the distributions under the user's control. A set of component values for each design is chosen randomly and inserted into the simulation. A subroutine is provided to convert the component nominal values and tolerances specified by the user into simulation parameters. Both

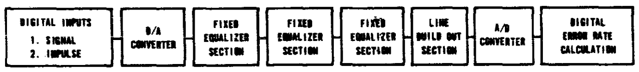


Figure 3—Hybrid equalizer simulation

an input corresponding to the pulse to be equalized and an impulse are sequentially applied to the perturbed design to evaluate intersymbol interference and noise power respectively. The analog outputs of the equalizer are sampled and processed on the digital computer to determine error rate. Even for this complex application, it is possible to evaluate the error rate for one design every five seconds.

In the subsequent discussion, only component tolerances in the three bridged-T sections are considered; it is assumed that these sections are buffered and all other parameters are set to their nominal values. All the components in the basic bridged-T equalizer shown in Figure 4 may have tolerances applied to them except the characteristic impedance labeled R. Care must be exercised in a tolerance analysis simulation to avoid programming simplifications that may not be valid when components are perturbed. For example, the bridged-T section can be analyzed as a second order system if it is assumed that the series and shunt arms are properly matched; but since this match would be destroyed by perturbations, a fourth order simulation is required.

Before the analytical difficulties introduced by a nonlinear criterion (error rate) are discussed, a linear criterion directly related to the equalizer response is discussed so that the differences can be illustrated.

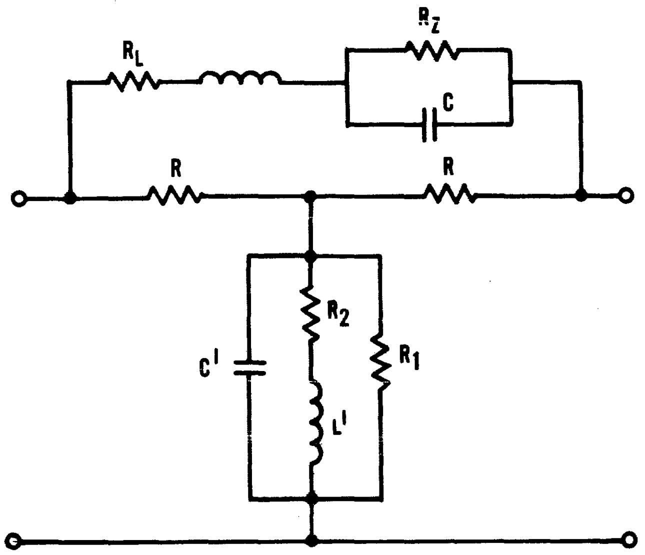


Figure 4—Bridged-T realization of fixed equalizer section

First, the standard deviation (σ) of the peak amplitude of the equalizer output pulse is estimated by the moment method for various sets of tolerances applied to the components in one bridged-T section. The same data are obtained by measurement with the tolerance analysis program and the results are compared. Then the error rate is measured under the same conditions and the influence of parameter tolerances are qualitatively compared.

The variance of the pulse peak can be estimated from the distributions applied to the component values by the moment method.[4] The performance criterion is expanded in a Taylor series and only the lowest order terms are used. Then the variance of the response (σ_p^2) is found for normally distributed, uncorrelated component perturbations with the formula.

$$\sigma_p^2 = \sum_{i=1}^{N} \left(\frac{\partial P}{\partial \alpha_i}\right)^2 \sigma_{\alpha_i}^2 \tag{1}$$

where the α_i are the component values. The partial derivatives of the response with respect to the parameters can be obtained by a finite difference approximation or by measurement using the parameter influence method.[5] The second method, which requires that an auxiliary circuit be programmed to measure the partials without an approximation, was used to obtain the data for the equalizer. The auxiliary circuit was realized by analog simulation and the measurements were sampled and normalized with a hybrid program to provide the sensitivity coefficients, S_i.

$$S_i = \frac{\partial \ln P}{\partial \ln \alpha_i} = \frac{\alpha_i}{P} \cdot \frac{P\partial}{\partial \alpha_i} \tag{2}$$

where P is the pulse peak with no components perturbations. Let us express the standard deviation of the component distributions about the nominal values in terms of tolerance specifications. We define the tolerance percentage (X) to correspond to the Kσ point of the normal distribution.
Thus

$$X = 100\left(\frac{K\sigma_{\alpha_i}}{\alpha_i}\right) \tag{3}$$

Equation (1) can be rewritten as:

$$\left(\frac{\sigma p}{P}\right)^2 = \sum_{i=1}^{N} S_i^2 \left(\frac{X}{100K}\right)^2 \tag{4}$$

This same additive procedure can be applied if perturbations are introduced in the component values of all three sections.

Equation (4) was used with measured sensitivity coefficients, S_i, to calculate the data given in Table I for various sets of component tolerances applied to one equalizer section. A comparison of the S_i revealed that peak amplitude is most sensitive to variations in C and L′. These two components have equal influence and no other component is even 1/20 as effective.

Table I—Standard deviation of pulse peak

Tolerance at 2σ (percent)			σ_p/P (10^{-2})	
R	*C*	*L*	Calculated with Equation (4)	Measured with Tolerance Program
0	0	0	0	0.03
1	1	1	0.56	0.50
1	1	3	1.23	1.20
3	3	3	1.68	1.80
5	5	5	2.80	2.71

To determine the σ of the pulse peak with the tolerance analysis program, the error rate calculation was replaced with a measurement of peak amplitude. The program was used to accumulate histograms, with at least 150 designs measured for each case, for the tolerance sets listed in Table I. The σ estimated from these histograms is also listed in Table I; the similarity of the results is obvious. The finite σ with no perturbations is due to small errors in the analog simulation; the program thus provides a check on analog reproducibility. The relative sensitivity of the components is determined with this program by experimentally varying tolerances and observing the histogram changes.

Let us now consider the influence of the parameters on the error rate under the same conditions listed in Table I. The histograms were clearly not normally distributed. These cumulative distributions were replotted on probability paper as shown in Figure 5. On this paper, a normal distribution appears as a straight line with σ inversely proportional to the slope. Since the distributions are not normal, the tolerance sets cannot be compared on the basis of their σ as in the moment method. The relative influence of the tolerance sets can be compared on the basis of yield at a selected rate as determined from the curves. The threshold of acceptability for this equalizer was an error rate less than 10^{-10}; therefore, using 1 percent tolerances with K = 2 for all components, approximately 98 percent of the designs are acceptable. With the low number of samples used to obtain this data, the absolute yields may not be

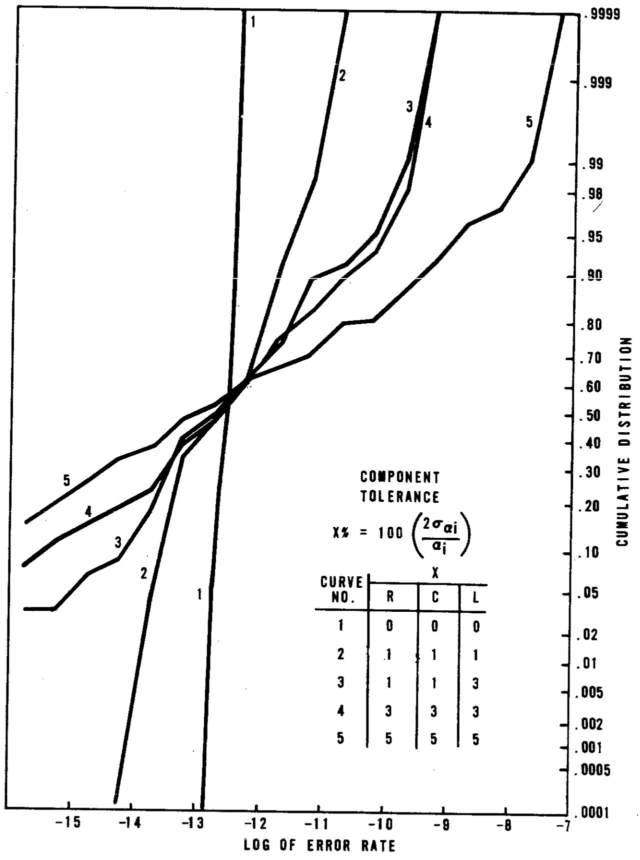


Figure 5—Influence of component tolerance on equalizer performance as determined by scaling the Gaussian distributions applied to the components in a single fixed section with various tolerance sets

extremely accurate but a comparison of relative values is valid. A comparison of curves two, three, and four in Figure 5 reveals that selectively tightening the tolerances on the R's and C's does reduce the variance, but that the yields are not appreciably improved until the L's are also tightened. With the linear criterion, it was observed that C and L′ were equally effective; but when comparing the yields for a given error rate, this is clearly not the case.

The difficulty of combining error rates is also evident from the curves of Figures 6 and 7 in which are compared the error rates of each of the three bridged-T sections perturbed separately and all three simultaneously perturbed. The simultaneous perturbation of all three did not produce performance appreciably different from the worst of the individual sections (each section had different nominal component values). If one attempts to combine the individual section data to obtain a worst case estimate of the overall three section performance, a much more pessimistic estimate would result than the performance actually measured.

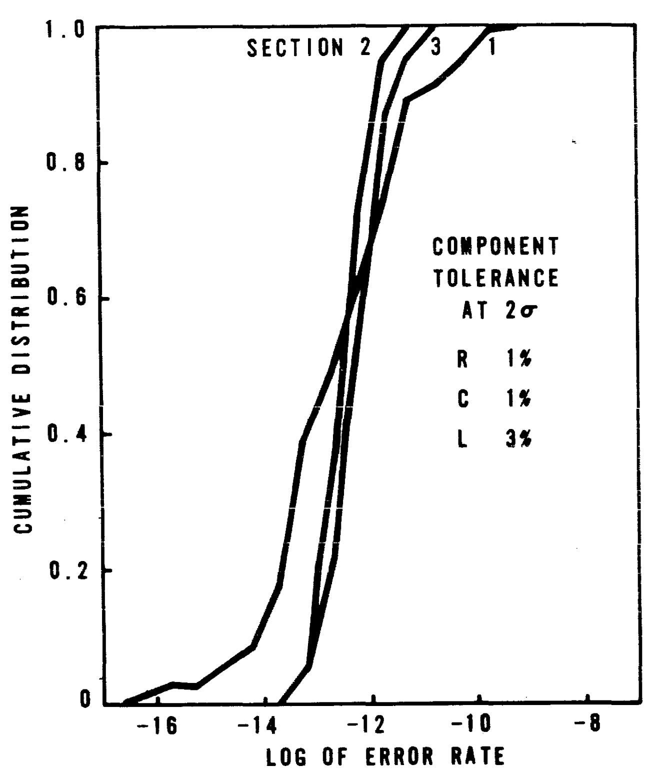


Figure 6—Relative influence of each of the three fixed sections in the equalizer as determined by applying Gaussian distribution to all components in the section

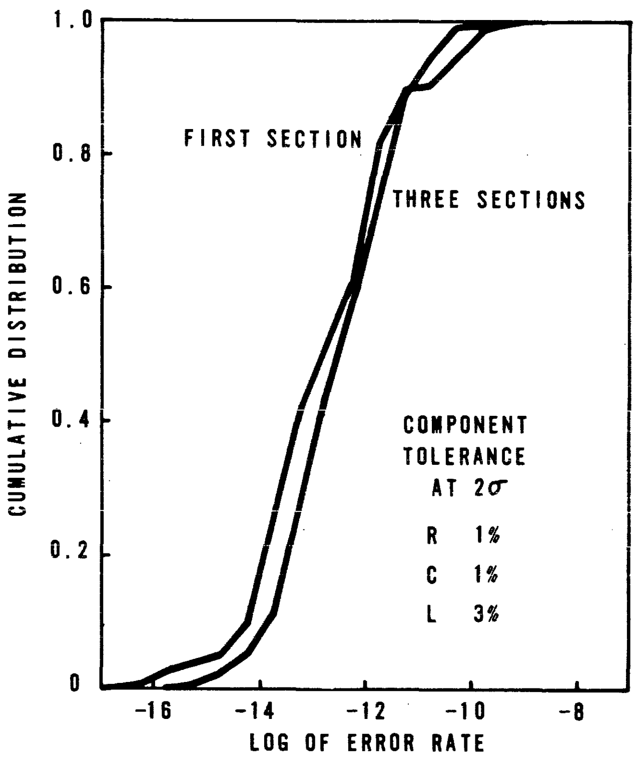


Figure 7—The dominant influence of the first of the three fixed sections of the equalizer is illustrated by applying Gaussian distribution to the components in only the first section and then to all three sections simultaneously

Pessimistic estimates of performance may also occur if fixed values on system parameters, other than components, are used in the calculation of error rate. For example, in obtaining the component tolerance data, a fixed value of crosstalk was used that was indicative of the worst cables measured. A statistical distribution that reflected the measured crosstalk in many cables was introduced into the error rate calculation by the program. Figure 8 shows a comparison of error rates for component variations in one section with fixed and with normally distributed crosstalk. It can be seen that the fixed value yielded an extremely pessimistic estimate since most designs will produce significantly better performance. If acceptable yields can be obtained with worst case estimates, there is no need for measuring the distributions. But typically with 'state of the art" designs this is not the case and the distributions may allow the designer to relax some specifications and still obtain adequate performance.

Realistic estimates of yield are invaluable in a cost comparison of different realizations. Rather than tighten all tolerance to provide a 100 percent yield and then compare relative costs, an alternative probabilistic approach can be taken. Since yields can be determined from the cumulative distributions, the cost can be compared on the basis of testing and rejecting a few of the units that are constructed with wider tolerance components. This cost calculation includes the yield to be expected, the cost (C_i) of the individual components with a specified tolerance (T_i) and the cost (C_T) of testing to determine defective units. Therefore, the cost of an acceptable unit, that is one with an error rate less than the threshold (k) is:

$$C = C_T + \frac{1}{M(T_1, \ldots T_N)} \sum_{i=1}^{N} C_i (T_i) \qquad (5)$$

N = number of components in a unit

$$C_T = \begin{cases} U \text{ if } M \neq 1 \\ O \text{ if } M = 1 \end{cases}$$

$M(T_1, \ldots T_N)$ is the yield at threshold k, for the specified set of component tolerances, as obtained from the cumulative distribution. This cost calculation is indicative of the many statistical applications in which the probability density or the cumulative distribution is needed to determine system performance.

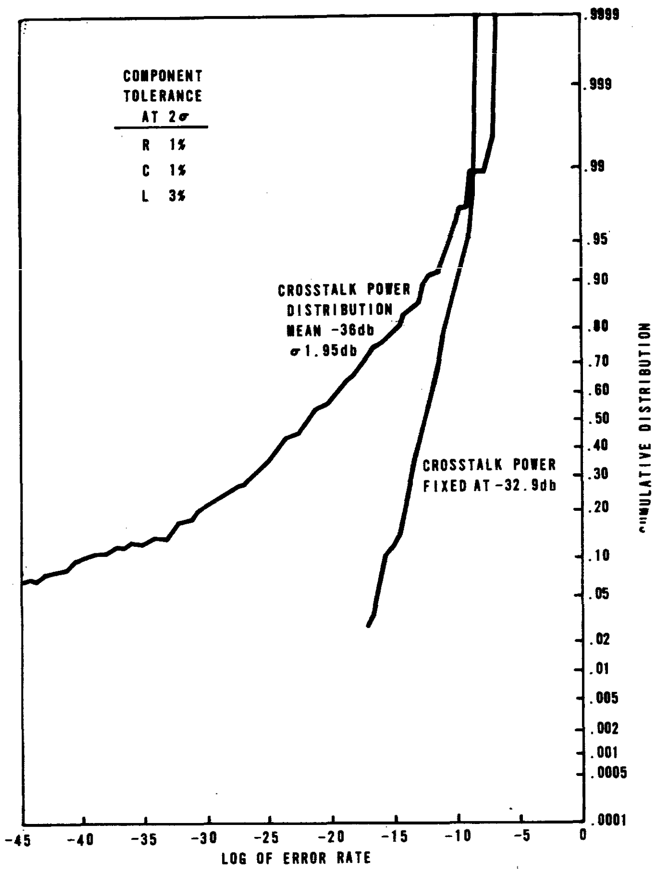


Figure 8—Comparison of equalizer performance as measured with the same Gaussian distributions applied to the components in the first fixed section using both fixed and Gaussianly distributed values for external crosstalk power

CONCLUSIONS

The values of the interactive computer with graphical display for tolerance analysis has been demonstrated in a variety of applications. The experimental approach provides the designer with realistic estimates of anticipated performance on which to base his decisions. The performance distributions are acquired economically because the user can quickly evaluate and terminate the computer runs.

This technique is to be made more readily available to designers by modifying a standard digital analysis program such as ECAP[6] to fit into the structure. Then a large class of circuit problems can be handled without special purpose programs and simulations being provided. Therefore, only the more difficult problems will require additional programming to perform the tolerance analysis.

ACKNOWLEDGMENT

The discussions of this material with J. Chernak, P. Fleischer, R. Snicer, and J. Davis are gratefully acknowledged. The author is particularly indebted to D. Bohling who organized and wrote the tolerance analysis program used in this work.

REFERENCES

1 *Special issue on computer-aided design*
Proc IEEE Vol 55 November 1967
2 P E FLEISCHER
Hybrid optimization for electrical circuit design
Eastern Simulation Council Princeton N J June 1968
3 D A CALAHAN
Computer-aided network design
McGraw-Hill New York 1968
4 D G MARKS L H STEMBER JR.
Variability analysis
Electro-Technology Vol 76 No 1 July 1965 37–48
5 H F MEISSINGER
The use of parameter influence coefficients in computer analysis of dynamic systems
Proc Western J C C 1960
San Francisco May 1960
6 *1620 electronic circuit analysis program (ECAP)*
User's manual H20–0170 IBM Report 1620–EE 02X

A method of automatic fault-detection test generation for four-phase MOS LSI circuits

by Y. T. YEN

National Cash Register Company
Dayton, Ohio

INTRODUCTION

To determine whether an integrated digital circuit is working properly, one may apply to the circuit a set of well-devised test sequences and compare the resultant outputs with the corresponding correct outputs. Any discrepancies indicate the presence of a fault. The main task here is to find a set of test sequences which can detect the presence of any prescribed fault in the circuit. This test generation problem will become formidable for large scale integrated arrays, since large number of logic circuits may be contained in an array with a limited number of exterior terminals.

Presently, the approach most commonly used in test generation is computer fault simulation. In other words, data are repetitively processed through a cycle of two steps, i.e., test generation followed by test verification. In each cycle, the generation of primary output test sequences can be accomplished by a computer logic simulation technique. The primary-input test sequences are manually generated by engineers because much creative work is involved.

In this paper, a method will be proposed to generate primary-input test sequences for 4-phase MOS sequential switching circuits. This method can find the shortest primary-input test sequences to detect a given fault on an array.

An MOS transistor stuck at either short or open permanently is the failure mode to be considered. A single fault assumption[4] will be made in this paper.

Peculiar circuit features

Four-phase MOS circuits exhibit many peculiar features not observed in other families of logic circuits.[2,3] It is found that the test sequence generation problem can be greatly simplified by exploiting the peculiar features of these circuits.

There are four clock signals and four types of logic gates in a four-phase MOS circuit.[2]

The time interval of one clock-phase is defined as one unit of time t, as shown in Figure 1. For example, t = 7 is the time at the end of phase 3 of the second bit.

Let S_t denote the set of logic levels of all internal logic gates of a 4-phase circuit at time t:

$$S_t = (s_{1t}, s_{2t}, s_{3t}, \text{—}, s_{it}, \text{—}, s_{mt})$$

Where s_{it} is the logic level of the i-th logic gate at time t, and m is the total number of internal logic gates.

It should be noted that the output signal of a 4-phase gate cannot be sampled during the certain clock-phase times, because of the precharging behavior of the gates. The clock-phase time during which the output signal of a gate can be sampled is shown in Table 1.

Table 1—Sampling time of gates

Type of gates	Clock-phase time during which the output signal of a logic gate can be sampled.
type−1	Phases 3 and 4
−2	Phase 4
−3	Phases 1 and 2
−4	Phase 2

Now the peculiar circuit features useful for test sequence generation can be summarized as follows:

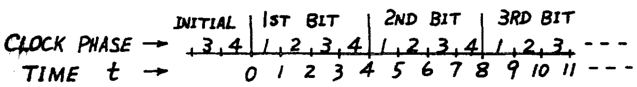


Figure 1—Definition of time t

(1) Each four-phase logic gate provides a storage of information at its output for one or two clock-phase times, (2) feedback and feed-forward in a flipflop circuit do not occur simultaneously, and (3) the logic levels of all logic gates in a circuit, including flipflops, can be initialized to predictable logic levels by inhibiting the occurrence of one ϕ_2 or ϕ_4 clock pulse.

Tracing technique for four-phase MOS circuits

In order to force the logic gates of a 4-phase circuit to desirable logic levels, we will trace a signal backward to find the necessary primary-input sequences and circuit initialization. The peculiar features of 4-phase circuits enable us to trace a 4-phase signal backward by considering each unit of time t as follows.

Let F_{t1} be the 4-phase signal at a logic gate output at time $t = t_1$. $\nabla^j F_{t1}$ will denote the function obtained from tracing F_{t1} backward by j units of time t, where j is a positive integer. To trace $\nabla^j F_{t1}$ backward by one unit of time, we will substitute every *internal* gate variable s_k of $\nabla^j F_{t1}$ with the combinatorial Boolean function implemented by the logic gate s_k. Let us demonstrate this technique by tracing backward the 4-phase signal s_1 in Figure 2.

In Figure 2, i_1, i_2, and i_3 are primary inputs. s_3 and s_4 are type-1 logic gates. s_2, s_1, and s_5 are type-2, 3, and 4 logic gates, respectively.

Since s_1 gate is a type-3 gate, the logic signal s_1 can be sampled only during phases 1 and 2. Let t_1 be the time at the end of phase 4 of bit N. Then $(t_1 - 1)$, $(t_1 - 2)$, $(t_1 - 3)$ and $(t_1 - 4)$ are the times at the end of phases 3, 2, 1 of bit N and phase 4 of bit N − 1, respectively.

Let s_{1tm} and i_{1tm} denote the logic level of s_1 and i_1 at time $t = t_m$, respectively. In tracing s_1 backward, the following relations are derived for each unit of time t:

$$s_{1t1} = \overline{s_{2(t1-1)}\cdot s_{4(t1-1)}} = \bar{s}_{2(t1-1)} + \bar{s}_{4(t1-1)} \quad (1)\ (2)$$

$$= s_{3(t1-2)} + \bar{s}_{4(t1-2)} \quad (3)$$

$$= \bar{i}_{1(t1-3)} + i_{2(t1-3)} + s_{1(t1-3)}\cdot s_{5(t1-3)} \quad (4)$$

$$= \bar{i}_{1(t1-3)} + i_{2(t1-3)} + \overline{(s_{2(t1-4)}\cdot s_{4(t1-4)})}\cdot \bar{i}_{3(t1-4)} \quad (5)$$

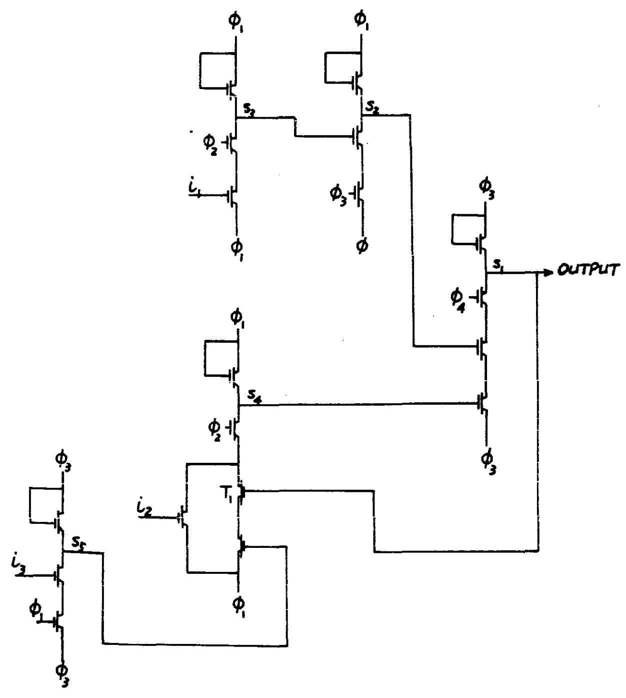


Figure 2—A four-phase MOS Circuit

$$= \bar{i}_{1(t1-3)} + i_{2(t1-3)} + (s_{3(t1-5)} + \bar{s}_{4(t1-5)})\cdot \bar{i}_{3(t1-4)} \quad (6)$$

Where (2) is derived by substituting s_1 of (1) with the combinatorial Boolean function $\overline{s_2 s_4}$, which is implemented by the logic gate s_1. Note that the time at which signal s_1 can be sampled is one unit later than that at which signals s_2 and s_4 can be sampled.

The relation (1) = (2) indicates that: (a) s_1 will be 1 at time $t = t_1$ if and only if either or both $s_2 = 0$ and $s_4 = 0$ hold at time $t = t_1 - 1$; and (b) s_1 will be 0 at time $t = t_1$ if and only if $s_2 = s_4 = 1$ holds at time $t = t_1 - 1$.

Let us denote s_{1t1} as F_{t1}. Then $\nabla^1 F_{t1}$ will be $s_{2(t1-1)}\cdot s_{4(t1-1)}$, as shown in (2).

Equation (3) is derived from (2) by tracing backward another unit of time t. There is a type-2 gate s_2 between gates s_3 and s_1. However, there is no type-2 gate between gates s_4 and s_1. Therefore, we will substitute s_2 of (2) with the combinatorial Boolean function $\bar{s}_3$ implemented by the logic gate s_2. And, we will not substitute s_4 of (2) at this time. $s_{3(t1-2)} + \bar{s}_{4(t1-2)}$ of (3) will be denoted as $\nabla^2 F_{t1}$.

Equation (4) is derived from (3) by tracing backward additional unit of time t. We substitute s_3 and s_4 of (3) with the combinatorial functions $\bar{i}_1$ and $\overline{i_2 + s_1 s_5}$, respectively. We will denote $\bar{i}_{1(t1-3)} + i_{2(t1-3)} + s_{1(t1-3)}\, s_{5(t1-3)}$ of (4) as $\nabla^3 F_{t1}$.

Equations (5) and (6) are similarly derived from (4) and (5), respectively.

It should be noted that gates s_1 and s_4 of Figure 2 form a RS flipflop. Recall that a 4-phase MOS circuit possesses a peculiar feature that the feedback and feedforward in a 4 phase flipflop cannot occur simultaneously. This feature enables us to trace backward a signal in a 4-phase sequential circuit for each unit of time, as shown above. No additional specification, such as a truth table, is needed to specify a flipflop.

Conditions of fault signal propagation

Let $\mathcal{f}(X_1, X_2, —, X_n)$ be the combinatorial Boolean function implemented by a 4-phase logic gate. *A fault of an input*, say X_i, will be defined as the MOS transistor fed by X_i is permanently stuck at short or open.

In this section, an algebraic method will be discussed to derive the input conditions of a logic gate under which the output of the logic gate will respond to the presence of a fault at an input X_i.

Boolean difference

See Figure 3. $\mathcal{f}(X_1, X_2, —, X_n)$ is a combinatorial Boolean function. Assume input X_m has a fault. Sellers et al.[1] show that the fault of X_m can be propagated to the output if and only if the input variables of $\mathcal{f}$, except X_m, satisfy the following condition:

$$\mathcal{f}(X_m \to 1) \oplus \mathcal{f}(X_m \to 0) = 1$$

Where $\oplus$ is exclusive OR, and m is one of the integers 1, 2, 3, —, n. $X_m \to 1$ (or 0) means that variable X_m of $\mathcal{f}$ is substituted by the constant 1 (or 0).

A Boolean difference of a function $\mathcal{f}_j(X_1, X_2, —, X_n)$ with respect to input X_m will be denoted as $D_{j,m}$, which is

$$D_{j,m} = \mathcal{f}_j(X_m \to 1) \oplus \mathcal{f}_j(X_m \to 0)$$

Examining the features of a 4-phase logic gate, we see that the fault of input X_m can be propagated to the output of the logic gate if and only if the inputs of the logic gate, except X_m, satisfy the following condition:

$$\mathcal{f}_j(X_m \to 1) \oplus \mathcal{f}_j(X_m \to 0) = 1$$

Where $\mathcal{f}_j$ is the combinatorial Boolean function implemented by the logic gate.

Let us consider the 4-phase gate shown in Figure 4. Assume that X_3 has a fault.
Then,

$$\mathcal{f}(X_3 \to 1) = \overline{X_1X_2 + X_4}$$
$$\mathcal{f}(X_3 \to 0) = \overline{X_1X_2}$$

Boolean difference

$$\begin{aligned} D_{1,3} &= \mathcal{f}_1(X_3 \to 1) \oplus \mathcal{f}_1(X_3 \to 0) \\ &= \overline{X_1X_2 + X_4} \oplus \overline{X_1X_2} \\ &= \bar{X}_1 X_4 + \bar{X}_2 X_4 \end{aligned}$$

Thus, $D_{1,3} = 1$ if and only if either $X_1 = 0$ and $X_4 = 1$ hold or $X_2 = 0$ and $X_4 = 1$ hold. Under either of these two input conditions this logic gate behaves as a logic inverter, i.e., $\mathcal{f}_1 = \overline{X_3}$. Consequently, the gate output $\mathcal{f}_1$ will then respond to any fault of X_3. It should be

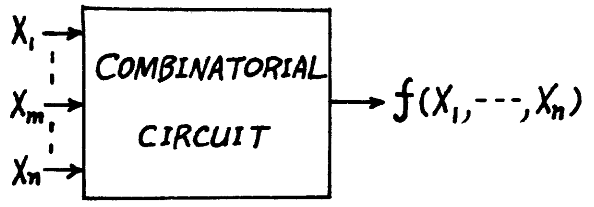


Figure 3—Signal notations of combinatorial circuit

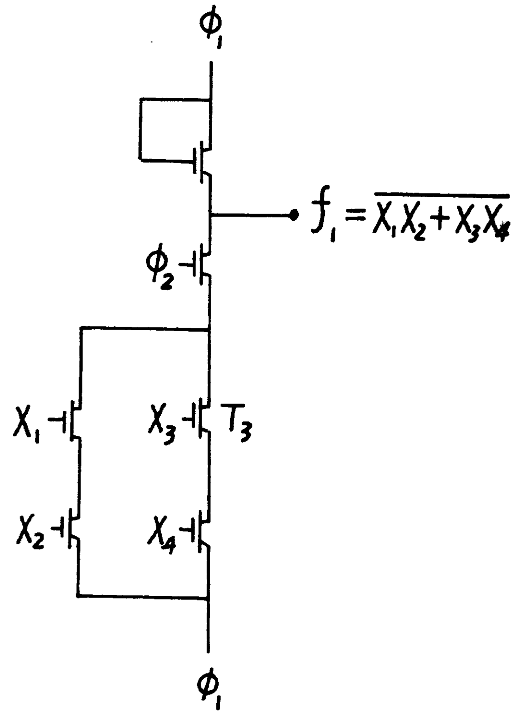


Figure 4—A four-phase MOS logic gate

noted that each term of $D_{j,i}$ represents such an input condition.

Assume the MOS transistor T_3 associated with input X_3 is permanently stuck at short. Then we must apply the signal $X_3 = 0$ to T_3, so that the fault of "short" can be detected at the output f_1. Therefore, the input conditions to detect T_3 permanently stuck at short are the input conditions satisfying $\overline{X}_3.D_{1,3} = 1$.
Where

$$\overline{X}_3.D_{1,3} = \overline{X}_3(\overline{X}_1X_4 + \overline{X}_2X_4) = \overline{X}_1\overline{X}_3X_4 + \overline{X}_2\overline{X}_3X_4\,.$$

In other words, either

$$X_1 = 0, X_3 = 0 \text{ and } X_4 = 1$$

or

$$X_2 = 0, X_3 = 0 \text{ and } X_4 = 1$$

will detect the fault of "short" of MOS transistor T_3.

If T_3 is permanently stuck at open, then $X_3.D_{1,3} = 1$ are the input conditions to detect such a fault.

The shortest fault detection test procedures

See the block diagram of a 4-phase circuit in Figure 5. Each block represents a 4-phase logic gate. Consider logic gate f_k. Assume the transistor associated with its input f_{k+1} has a fault. We will find the shortest tests to detect this fault. The propagation path of the fault is f_k, f_{k-1}, —, f_3, f_2, and f_1. Starting from f_1, we will find the Boolean difference and trace backward for each stage of logic gates until we reach f_{k+1} as follows.

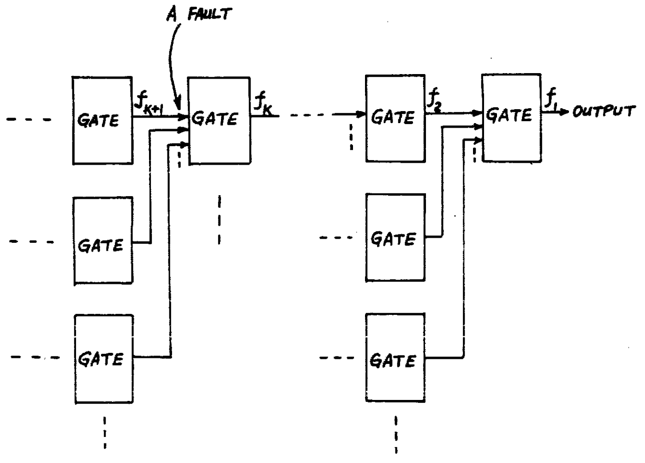


Figure 5—Block diagram of a four-phase circuit (Sequential or combinatorial)

We will denote functions $g_{j,j+1}$, for $1 \leq j \leq k$, as follows:

$$\begin{aligned} g_{1,2} &= D_{1,2} \\ g_{2,3} &= (\nabla^b g_{1,2})\cdot D_{2,3} \\ g_{3,4} &= (\nabla^b g_{2,3})\cdot D_{3,4} \\ g_{4,5} &= (\nabla^b g_{3,4})\cdot D_{4,5} \\ &--- \quad ----- \\ g_{k,k+1} &= (\nabla^b g_{k-1,k})\cdot D_{k,k+1} \end{aligned}$$

Where $D_{j,j+1}$ is the Boolean difference of f_j with respect to f_{j+1}. ∇^b is to trace $g_{j,j+1}$ backward b units of time t. b = 1 if one of f_j and f_{j+1} gates is either type-2 or type-4 logic gate. b = 2 if both f_j and f_{j+1} gates are neither type-2 nor type-4 logic gates.

Let us consider the gate f_k. If the transistor associated with input f_{k+1} is permanently stuck at short, $\overline{f}_{k+1}\cdot g_{k,k+1} = 1$ will be the input conditions to detect the prescribed fault. If the transistor associated with input f_{k+1} is permanently stuck at open, $f_{k+1}\cdot g_{k,k+1} = 1$ will be the input conditions to detect the prescribed fault. We will denote

$G = \overline{f}_{k+1}\, g_{k,k+1}$ if the transistor associated with f_{k+1} is stuck at short,

$= f_{k+1}\, g_{k,k+1}$ if the transistor associated with f_{k+1} is stuck at open.

Every term of G is an input condition to detect the prescribed fault. Each term of G may consist of primary inputs and internal logic gate signals. We will check each term of G to find whether it satisfies the following condition:

Condition A:

either
1. All variables of a term of G are primary inputs

or
2. All internal-gate variables of a term of G will become logic-1 by inhibiting a proper clock signal.

We will follow the following steps to find the shortest fault detection tests.

Step 1. Check every term of G to determine if it satisfies condition A.

Step 2. If no term of G satisfies condition A, trace G backward by one unit of time t and then repeat step 1. If there is a term

> of G which satisfies condition A, this term is the shortest test to detect the prescribed fault. Furthermore, if there exists a test at time t = t_r which can detect the prescribed fault, then at least one term of G at time t = t_r satisfies condition A.

If the function G becomes 0 at some step of tracing backward, then there exists no test which can detect the prescribed fault.

Let us derive the shortest tests to detect some fault in the circuit shown in Figure 2. Assume T_1 is stuck at open permanently. Then $\mathscr{f}_1$ and $\mathscr{f}_2$ are the gates s_1 and s_4, respectively. Since the faulty transistor T_1 belongs to gate s_4, $\mathscr{f}_k$ and $\mathscr{f}_{k+1}$ are s_4 and s_1, respectively

Let t_1 denote the reference time at s_1.
Since

$$\mathscr{f}_1 = s_{1(t1)} = \overline{s_{2(t1-1)}\, s_{4(t1-1)}}$$

$$D_{1,2} = \mathscr{f}_1(s_{4(t1)} \rightarrow 1) \oplus \mathscr{f}_1(s_{4(t1)} \rightarrow 0)$$

$$= s_{2(t1-1)}$$

thus

$$g_{1,2} = s_{2(t1-1)}$$

Since

$$\mathscr{f}_2 = s_{4(t1-2)} = \overline{i_{2(t1-3)} + s_{1(t1-3)}\, s_{5(t1-3)}}$$

$$D_{2,3} = \mathscr{f}_2(s_{1(t1-3)} \rightarrow 1) \oplus \mathscr{f}_2(s_{1(t1-3)} \rightarrow 0)$$

$$= s_{5(t1-3)} \cdot \overline{i_{2(t1-3)}}$$

$$\nabla^2 g_{1,2} = i_{1(t1-3)}$$

thus,

$$g_{2,3} = s_{5(t1-3)} \cdot i_{1(t1-3)} \cdot \overline{i_{2(t1-3)}}$$

Since T_1 is stuck at open,

$$G = \mathscr{f}_3 \cdot g_{2,3}$$

$$= s_{1(t1-3)}\, s_{5(t1-3)}\, i_{1(t1-3)}\, \overline{i_{2(t1-3)}}$$

We will check G to determine if it satisfies Condition A. Since internal-gate variables s_1 and s_5 cannot become logic-1 simultaneously at time ($t_1 - 3$) by inhibiting either ϕ_2 or ϕ_4 clock signal, we will trace G backward one unit of time t. Here, it should be noted that gate s_5 is a type-4 gate and gate s_1 is a type-3 gate.

$$\nabla^1 G = s_{1(t1-4)} \cdot \bar{i}_{3(t1-4)} \cdot i_{1(t1-3)} \cdot \bar{i}_{2(t1-3)}$$

Let us check ∇^1G to determine if it satisfies Condition A. If we initialize the circuit by inhibiting ϕ_4 at time $t_1 - 5$, then all type-3 gates such as the gate s_1, will become logic-1 at time ($t_1 - 4$) and ($t_1 - 3$); therefore, ∇^1G satisfies condition A. In other words, if we set up the following condition, the fault of T_1 being stuck at open can be detected at the primary output s_1 at time t = t_1:

Inhibiting ϕ_4 at time ($t_1 - 5$)

$$i_3 = 0 \text{ at time } (t_1 - 4)$$

$$i_1 = 1 \text{ at time } (t_1 - 3)$$

$$i_2 = 0 \text{ at time } (t_1 - 3)$$

Since ∇^1G has only one term, this condition is the only test to detect the fault of T_1 being stuck at open.

Computer programming

Manual implementation of this method was not attempted for the complexity of logic in an LSI array because of the complex detail involved. Using this method, a computer program written in FORTRAN IV was developed and has been used to generate test procedures for each prescribed fault in a 4-phase MOS LSI array.

This computer program consists of 15 subroutines to perform the following manipulation:

Simplify a function
OR functions
AND functions
Complement a function
Find Boolean difference
Trace a function backward
Check a function to determine if it satisfies Condition A
Simulate faulty signals
Keep tracking timing.

Every signal name is denoted by a number. A Boolean function F_x is specified by an integer number NXTERM, and two integer arrays NXNOVAR(I) and NXVAR(I). For example, $F_x = s_1s_3s_4 + s_2\bar{s}_4$ and $F_y = \bar{s}_2s_5 + s_6$ will be specified as follows in the computer program:

F_x: NXTERM = 2
NXNOVAR(1) = 3, NXNOVAR(2) = 2
NXVAR(1) = 1, NXVAR(2) = 3,
NXVAR(3) = 4
NXVAR(4) = 2, NXVAR(5) = −4.

F_y: NYTERM = 2
NYNOVAR(1) = 2, NYNOVAR(2) = 1
NYVAR(1) = −2, NYVAR(2) = 5
NYVAR(3) = 6

Where NXTERM specifies the number of terms in function F_x. NXNOVAR(1) and NXNOVAR(2) show the number of variables in the first and second terms. NXVAR(i) shows the variable at the *ith* order counting from the left of the function F_x, i.e., NXVAR(1) = 1 denotes s_1, and NXVAR(5) = −4 denotes $\bar{s}_4$.

F_y is similarly specified in the program.

Now, let us OR the functions F_x and F_y. The resultant function $F_z = F_x + F_y$ can be easily performed by the computer program as follows.

F_z: NZTERM = NXTERM + NYTERM = 4
NZNOVAR(1) = 3, NZNOVAR(2) = 2
NZNOVAR(3) = 2, NZNOVAR(4) = 1
NZVAR(1) = 1, NZVAR(2) = 3,
NZVAR(3) = 4
NZVAR(4) = 2, NZVAR(5) = −4,
NZVAR(6) = −2
NZVAR(7) = 5, NZVAR(8) = 6.

Where F_z denotes $s_1s_3s_4 + s_2\bar{s}_4 + \bar{s}_2s_5 + s_6$.

Input data

The input data to this computer program only consist of:

a. Logic implemented at every individual logic gate. This logic is a combinatorial Boolean function. For example, the program needs the combinatorial Boolean function $\overline{s_2 + s_4}$ to specify the logic implemented by s_1 gate in Figure 2.
b. Type of each logic gate. s_1 gate in Figure 2, for example, is a type-3 gate.
c. Designation of faulty MOS transistors.
d. Designation of one fault-signal propagation path for each faulty gate. An internal signal of an LSI circuit may be propagated to several primary outputs. To simplify the computer program, a desirable propagation path for an internal fault signal need be given. The propagation path of a fault signal s_4 in Figure 2, for example, will be specified as $s_4 \rightarrow s_1$ in the input data, for the faults associated with s_4 gate.

This computer program is written in FORTRAN IV for a NCR 315 RMC computer. Through the use of 40K-word core memory, 12 bits per word, it can generate the primary input sequences of the shortest fault detection tests for each prescribed fault in a 4-phase MOS array of 70 four-phase logic gates. The program needs 8 minutes of FORTRAN IV NCR 315 RMC computer time for the prescribed faults of *one 3-input four-phase gate* on a typical 70-gate array. The computer time for this test generation can be greatly reduced if this FORTRAN program is converted into a NEAT (machine) language version for the NCR 315 RMC computer.

REFERENCES

1 F F SELLERS JR M Y HSIAO L W BEARNSON
Analyzing errors with the Boolean difference
Digest of the First Annual IEEE Computer Conference September 1967 6–9

2 Y T YEN
A mathematical model characterizing four-phase MOS circuits for logic simulation
IEEE Transactions on Computers Vol C–17 September 1968 822–826

3 Y T YEN
Intermittent failure problems of four-phase MOS circuits
To be published on IEEE Journal of Solid-State Circuits

4 E R JONES C H MAYS
Automatic test generation methods for large scale integrated logic
IEEE Journal of Solid-State Circuits Vol SC–2 No 4 December 1967

5 J P ROTH
Diagnosis of automata failures: A calculus and a method
IBM Journal July 1966 278–291

6 E G MANNING H Y CHANG
A comparison of fault simulation methods for digital systems
Digest of the First Annual IEEE Computer Conference September 6–8 1967 10–13

7 F P PREPARATA G METZE R T CHIEN
On the connection assignment problem of diagnosable system
IEEE Trans on Electronic Computers Vol EC–16 December 1967 848–854

8 S SESHU D N FREEMAN
The diagnosis of asynchronous sequential switching systems
IRE Trans Electronic Computers Vol EC–11 August 1962 459–465

A method of diagnostic test generation

by A. B. CARROLL, M. KATO,* Y. KOGA, and K. NAEMURA

University of Illinois
Urbana, Illinois

INTRODUCTION

A variety of diagnostic techniques[1-8] have been proposed and applied to error detection and location in computers. The efficiency of the tests generated by them varies from machine to machine depending on the scale of the system, its logic organization, and the employed hardware technology. The impact of highly integrated circuitry is changing the trend in logical design of computers.

The significant reduction of propagation delay per individual switching gate and of the physical size of the circuits has made it possible and frequently even desirable to use a less sophisticated technique in the logic design. The use of adder packages with standard carry lookahead and semiconductor scratch pad memories often results in a better cost/performance ratio than development of the fastest carry propagation technique and the most advanced instruction lookahead control.

The design of the ILLIAC IV computer[9] is one of the recent examples in which modularity is preferred to an excessive sophistication of logic.

On the other hand, the introduction of higher integration technique has posed a problem to diagnostic engineers. The failure modes of the circuits have become more complicated and less obvious. Seventy to eighty percent of catastrophic errors may still be caused by mechanical failures at bonding and connections but the other errors include such subtle faults as marginal errors rising from relatively low noise margins of current mode gates and unexpected shorts (low resistance) between logically distant connections on a semiconductor chip.

This paper describes a new approach to the method of diagnostic test generation for a computer built with highly integrated circuitry.

* Currently with Nippon Telegraph and Telephone Public Corporation, Tokyo, Japan.

The first half of the next section describes the generation of test paths used to test registers and transfer paths between them. The second half is concerned with generation of input patterns for testing combinational logic networks.

Techniques employed in logic simulation and location of errors are briefly discussed in the following sections.

Generation of test cases

Path tests

A graph representation of a system is useful to visualize the operation and data flows in it. Several papers have been published on the use of graphs in diagnosis of logical machines.[6] In this section, we discuss the generation of efficient tests for detection and location of errors using a graph of the machine.

Let us assume that the system has been represented as a directed graph, in which nodes and arcs correspond to some circuit blocks and signal lines. In addition, let us assume the arcs in the graph can be enabled or disabled independently. A graph is equivalently represented as a square matrix called "adjacency matrix."

Figure 1 shows an example of a graph and its adjacency matrix is as follows:

$$\begin{array}{c|cccccc} & N_0 & N_1 & N_2 & N_3 & N_4 & N_5 \\ \hline N_0 & 0 & 1 & 1 & 1 & 0 & 0 \\ N_1 & 0 & 0 & 0 & 1 & 0 & 1 \\ N_2 & 0 & 0 & 0 & 1 & 0 & 0 \\ N_3 & 0 & 0 & 0 & 0 & 1 & 1 \\ N_4 & 0 & 0 & 1 & 0 & 0 & 1 \\ N_5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

There exist thirteen possible paths from the input node N_0 to the output node N_5 as shown in Figure 2.

Let us define a row vector of Boolean elements for each path, where each column denotes a node or an arc; 0 indicates the absence of the corresponding node or arc in this path. Now we have the following matrix with respect to the graph in Figure 1. We call it path matrix of the graph.

$$
\begin{array}{c|cccccc cccccccccc|}
 & N_0 & N_1 & N_2 & N_3 & N_4 & N_5 & A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 & A_{10} \\
P_0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
P_1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
P_2 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
P_3 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
P_4 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
P_5 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
P_6 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
P_7 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
P_8 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
P_9 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
P_{10} & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
P_{11} & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
P_{12} & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{array}
$$

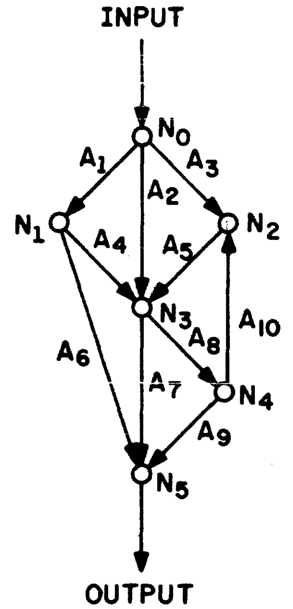


Figure 1—Example of a directed graph

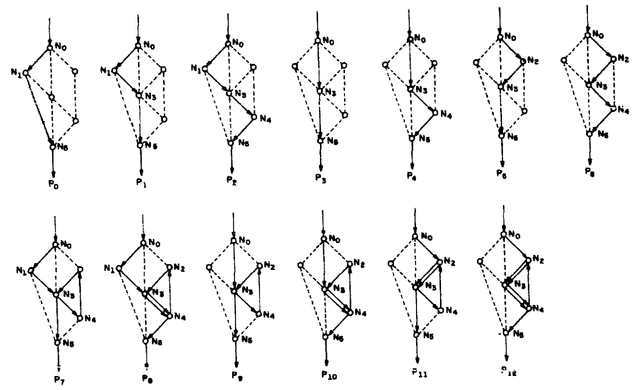


Figure 2—Possible paths

To diagnose a machine for disconnection errors, we should test paths to see if they successfully connect the input and output. In the above example, if arc A_9 fails, P_0, P_1, P_3, P_5, P_7, P_9, and P_{11} still connect successfully, but P_2, P_4, P_6, P_8, P_{10}, and P_{12} contain an error. Thus, by comparing columns of the path matrix with the test result, we can determine which node(s) and/or arc(s) have errors.

If an error location can be determined by testing all possible paths, it is said to be *distinguishable*. All possible path tests may not be required to determine where a single fault has occurred; i.e., we may be able to reduce the number of paths from the complete set of all possible paths in the original graph.

The following method may be used to reduce the number of path tests; the resulting set of paths still retain the ability to detect single failures.

Let us consider a graph with a single input and a single output; if there are multiple inputs or outputs, we can add a dummy input or output node and connect it to the actual inputs or outputs so that the graph may have only one input and one output. Now we produce a tree from the original graph; starting at the input node, we put down all the nodes which are directly fed by the input node and draw lines corresponding to the arcs between them. Level zero is assigned to the input node and level one to the nodes adjacent to the input node. The level one nodes of the example in Figure 1 are N_1, N_2, and N_3.

This step is repeated until all nodes are covered. If a node has already occurred on a higher level or previously on this level, we define it as a pseudo-terminal node and cease to trace arcs down from it.

Three pseudo-terminal nodes on level two are shown in Figure 3. Whenever a path from the input has met a pseudo-terminal node, we choose one of the routes which go down to the output to complete the path. We obtain six paths from the graph in Figure 1, as shown in Figure 4. In the example the shortest path was selected after the pseudo-terminal nodes.

Errors in node N_0 and N_5 are indistinguishable. So are the errors in N_2 and arc A_5, and those in N_1 and A_1, those in N_4 and A_8, respectively. This was true also in the original graph.

We call this method as the *p*ath *g*enerating *m*ethod hereafter in this paper, and we show that a set of paths generated by the PGM is a minimal and sufficient set of paths to detect and locate a failure in a network without feedback.

Theorem 1. The set of paths generated by the PGM is a minimal set that is sufficient for detecting and locating any distinguishable single failure within the graph without feedback.

Proof. If we insert a dummy node on each arc of the original graph, we can discuss only the distinguishability of node failures. The indistinguishable nodes are defined as follows: if there exist two nodes n_i and n_j such that no path in the graph passes through exactly one of the two nodes n_i and n_j, then these two nodes are said to be indistinguishable. It is noted that all nodes of the original graph are exhaustively included in the paths generated by the PGM.

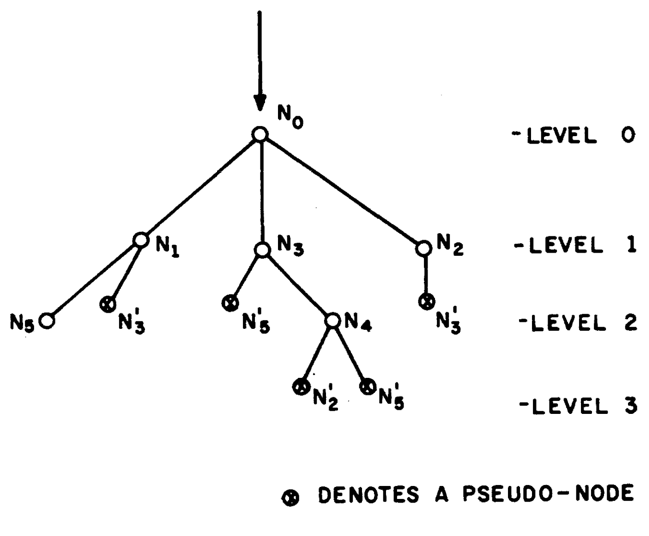


Figure 3—Generation of a tree from the graph (1)

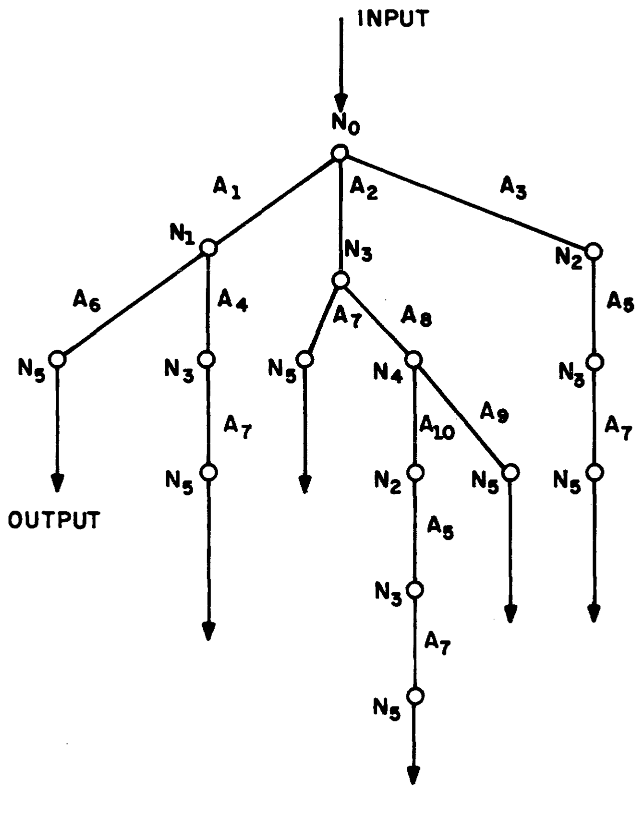


Figure 4—Generation of a tree from the graph (2)

$$\begin{array}{cccccc|cccccccccc}
N_0 & N_1 & N_2 & N_3 & N_4 & N_5 & A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 & A_{10} \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}$$

Figure 4.2—Path Matrix

If any one of the paths generated by the PGM were deleted, then there would exist at least one node (corresponding to an arc in the original graph) which was not included in the set of paths. This is obvious because no arcs are repeated in a tree generated by the method except in the route below the pseudo-terminal nodes.

The indistinguishable nodes resulting from the testing of all paths in the original graph do not become distinguishable nodes by testing the paths generated by PGM.

Next, let us assume that the nodes n_i and n_j are distinguishable by testing all paths with the original graph. There should be at least one path which passes through either one of the two nodes n_i and n_j. Let us define a set $\{p_{ij}\}$ of all possible paths that pass through both nodes n_i and n_j, and define a set $\{n_{ij}\}$ of all nodes on these paths lying between n_i and n_j, where the set $\{n_{ij}\}$ includes nodes n_i and n_j. Because n_i is distinguishable from n_j, there is at least one branch connected between node $n_k \notin \{n_{ij}\}$ and a node $n_\ell \in \{n_{ij}\}$. The path through the node n_ℓ exists in a set of paths generated by the PGM and this path does not pass through both nodes n_i and n_j. Thus the nodes n_i and n_j are still distinguishable by testing the paths generated by the PGM.

The time-consuming procedure of reducing the set of tests using a minimal cover technique can be avoided by the simple algorithm and the theorem stated above. The theorem is valid for a general class of networks with feedback.

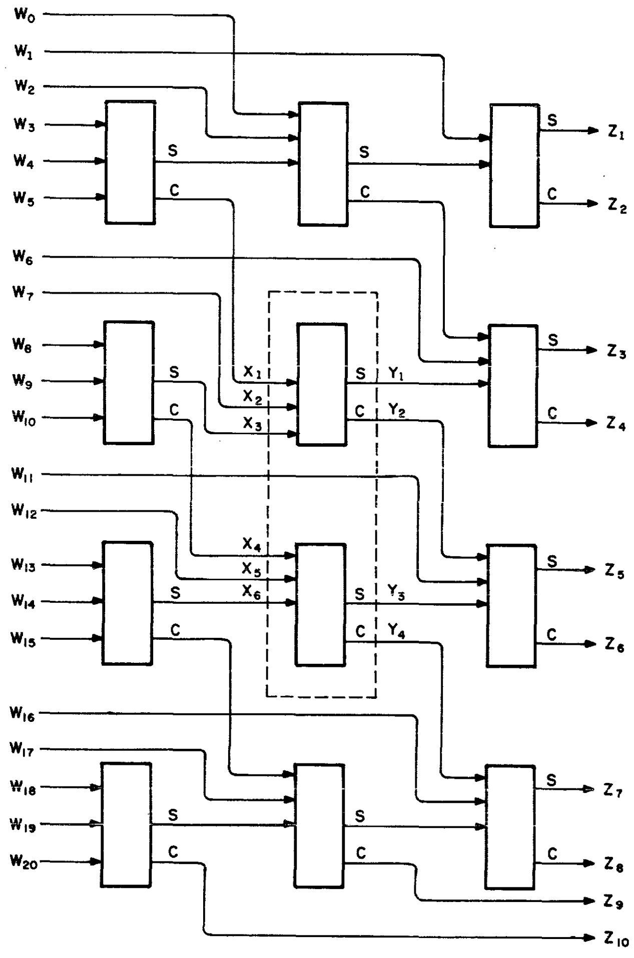


Figure 5—Example of a combinational network

Combinational tests

Suppose each solid box in the network of Figure 5 is a full adder and every pair of circuits is put on a chip as indicated by the dashed line. Let us consider how we can test the logic function of a chip.

The following equations are relevant to the chip inputs and outputs:

1. Equations for chip input signals

$$\left.\begin{aligned} x_1 &= w_3w_4 + w_4w_5 + w_3w_5 \\ x_2 &= w_7 \\ x_3 &= w_8 \oplus w_9 \oplus w_{10} \\ x_4 &= w_8w_9 + w_9w_{10} + w_8w_{10} \\ x_5 &= w_{12} \\ x_6 &= w_{13} \oplus w_{14} \oplus w_{15} \end{aligned}\right\} \quad (1)$$

2. Equations for chip function

$$\left.\begin{aligned} y_1 &= x_1 \oplus x_2 \oplus x_3 \\ y_2 &= x_1x_2 + x_2x_3 + x_1x_3 \\ y_3 &= x_4 \oplus x_5 \oplus x_6 \\ y_4 &= x_4x_5 + x_5x_6 + x_4x_6 \end{aligned}\right\} \quad (2)$$

3. Equations for external outputs

$$\left.\begin{aligned} z_3 &= (w_0w_2 + (w_0 + w_2)(w_3 \oplus w_4 \\ &\quad \oplus w_5)) \oplus w_6 \oplus y_1 \\ z_4 &= (w_0w_2 + (w_0 + w_2)(w_3 \oplus w_4 \\ &\quad \oplus w_5))(w_6 + y_1) + y_1w_6 \\ z_5 &= w_{11} \oplus y_2 \oplus y_3 \\ z_6 &= w_{11}y_2 + y_2y_3 + w_{11}y_3 \\ z_7 &= (w_{13}w_{14} + w_{14}w_{15} + w_{13}w_{15}) \\ &\quad \oplus w_{17} \oplus w_{18} \oplus w_{19} \oplus w_{20} \\ &\quad \oplus w_{16} \oplus y_4 \end{aligned}\right\} \quad (3)$$

In general, an output of a chip may feed an input to another chip whose output feeds an input (possibly through a few other chips) to the first chip. The equations for chip inputs in such a case will contain a term including the chip output variable.

Thus we define a "combinational logic network with pseudo-feedback loops" as follows:

1. It has a (multi-output) *combinational logic func-*

tion which describes the relation between the external inputs ($w_1, w_2, \ldots, w_N$) and outputs ($z_1, z_2, \ldots, z_M$).

2. It is composed of a finite number of unit circuits each of which has a (multi-output) combinational logic function to describe the relation between its inputs ($x_1, x_2, \ldots, x_n$) and outputs ($y_1, y_2, \ldots, y_m$).
3. It may contain pseudo-feedback loops; i.e., its connection graph may contain any number ot loops provided that none of these constitute storage of internal states. Thus, the output values of any unit circuit are uniquely determined by the external input values of the network despite the existence of loops.

The unit circuits in the network may be of any size and any structure, provided they satisfy premise (2) above. Specifically, however, we are concerned with unit circuits as a model of integrated circuit chips or portions of chips, the total number of input and output lines being limited within the order of 10^1. The structure of the circuit may be a net of at most $10^{1\sim2}$ of elementary logic gates.

Various types of failures may exist within the logic network. Some of them are of electrical nature, others of mechanical. A considerable fraction of them are only intermittent and cannot be detected except by chance, while the rest persist long enough to be detected as permanent failures.

It has been a convention in previous studies to classify those failures by logical modes: stuck-at-0 mode, stuck-at-1 mode, other degenerate modes, and non-degenerate modes. Very few papers have treated all of these modes. Techniques for detection and location of more than one failure in a chip have not been extensively developed.

So that we may not confine possible failures to any particular mode, we set the following assumptions, which have been adopted (if not explicitly) by all previous studies:

1. The failure is *non-sequential*; i.e., it may be any transformation of the combinational logic function of a unit circuit but it doesn't change the unit to a sequential circuit.
2. Only *critical* failures are of concern; namely, only if they change the external logical function of the network.
3. We will treat failures in any *single chip*; i.e., we assume that failures don't exist simultaneously in more than one chip, although the existence of more than one failure in a chip is not excluded.

The assumption (1) above requires that every possible combination of input values be applied to a unit circuit. (In case a subset of all possible combinations is obtained with a narrower assumption on failure modes, the following discussion can be applied to the subset.)

Then, the problem is how to determine external input values so that a unit circuit can be given an arbitrary input combination and that any change in its output value can be detected at an external output. This is formulated as follows.

First, obtain the equation f_i for a chip input x_i (cf. Figure 6), as follows:

1. Break up all the chip output lines, $v_1, \ldots, v_m$.
2. Insert dummy inputs $y_1, y_2, \ldots, y_m$ to the network to replace signals $v_1, \ldots, v_m$.
3. Write an equation f_i^* for x_i in terms of external inputs $w_1, \ldots, w_N$ and dummy signals $y_1, \ldots, y_m$.

Now the requirement is stated in the following, where the Boolean difference[7] $d\mu/dt_i$ of a function $\mu = \mu(t_1, \ldots, t_n)$ with respect to t_i is defined by $\mu(t_1, \ldots, t_n) \oplus \mu(t_1, \ldots, t_{i-1}, \bar{t}_i, t_{i+1}, \ldots, t_n)$.

1. Any combination $a_1, a_2, \ldots, a_n$ of values may be assigned to chip inputs $x_1, x_2, \ldots, x_n$,; i.e., for $i = 1, \ldots, n$.

$$f_i^*(w_1, \ldots, w_N, v_1, \ldots, v_m) = a_i \tag{4}$$

where v_j is an output from an error-free copy of the chip under test and is a function of $w_1, \ldots, w_N$ as obtained from the original network (Figure 6 (a)).

2. Some of the external output values should be changed by a deviation in signals $y_1, \ldots, y_m$. Thus, for $j = 1, 2, \ldots, m$:

$$\frac{dz_1}{dy_j} + \frac{dz_2}{dy_j} + \ldots + \frac{dz_M}{dy_j} = 1 \tag{5}$$

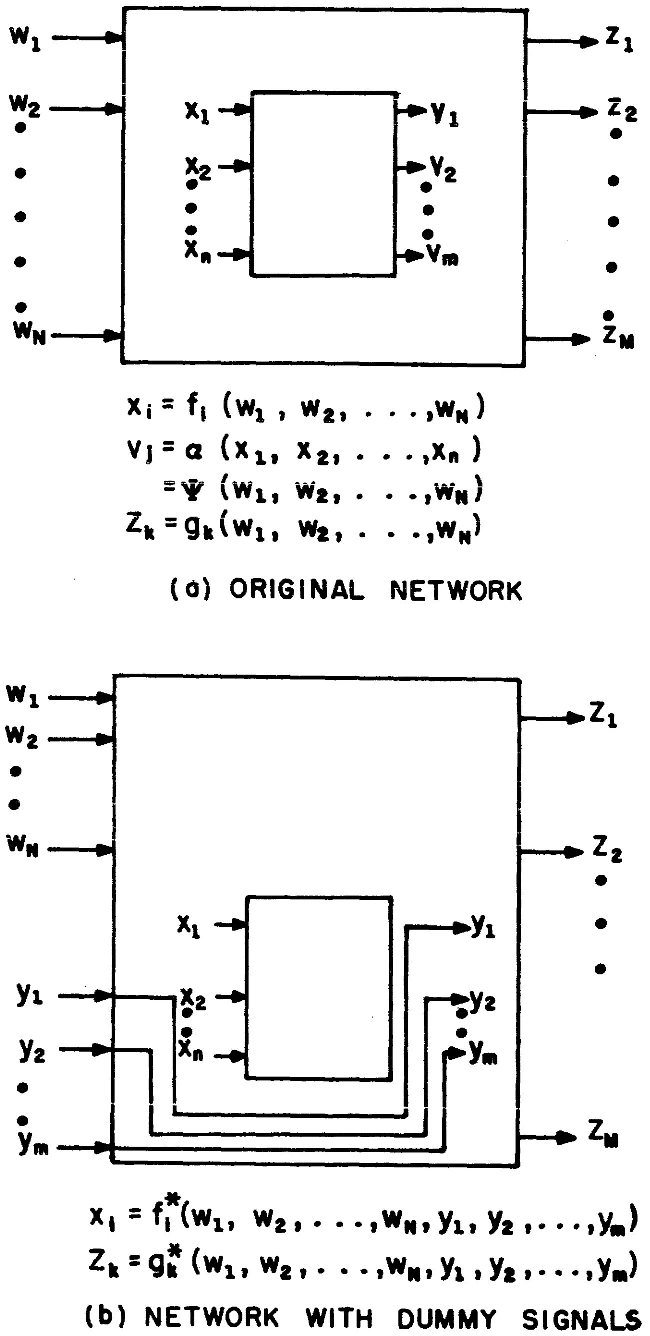
3. Any pseudo-feedback loops should be logically broken up to fix the chip input values in the existence of an error.

For: $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$

$$\frac{dx_i}{dy_j} = 0 \tag{6}$$

Thus, solving equations (4)—(6) in terms of $w_1, w_2, \ldots, w_N$ we obtain a set of input values for error detection.

A solution for the example in Figure 5 and equations (1)–(3) is shown in Figure 7. Sixty-four input patterns are required to test for all possible combinations of inputs to the chip.



W_1
W_2
W_N
Z_1
Z_2
Z_M
X_1
X_2
X_n
Y_1
Y_2
Y_m
$x_i = f_i(w_1, w_2, \ldots, w_N)$
$v_j = \alpha(x_1, x_2, \ldots, x_n) = \Psi(w_1, w_2, \ldots, w_N)$
$Z_k = g_k(w_1, w_2, \ldots, w_N)$
(a) ORIGINAL NETWORK
$x_i = f_i^*(w_1, w_2, \ldots, w_N, y_1, y_2, \ldots, y_m)$
$Z_k = g_k^*(w_1, w_2, \ldots, w_N, y_1, y_2, \ldots, y_m)$
(b) NETWORK WITH DUMMY SIGNALS

Figure 6—General representation of a combinational network

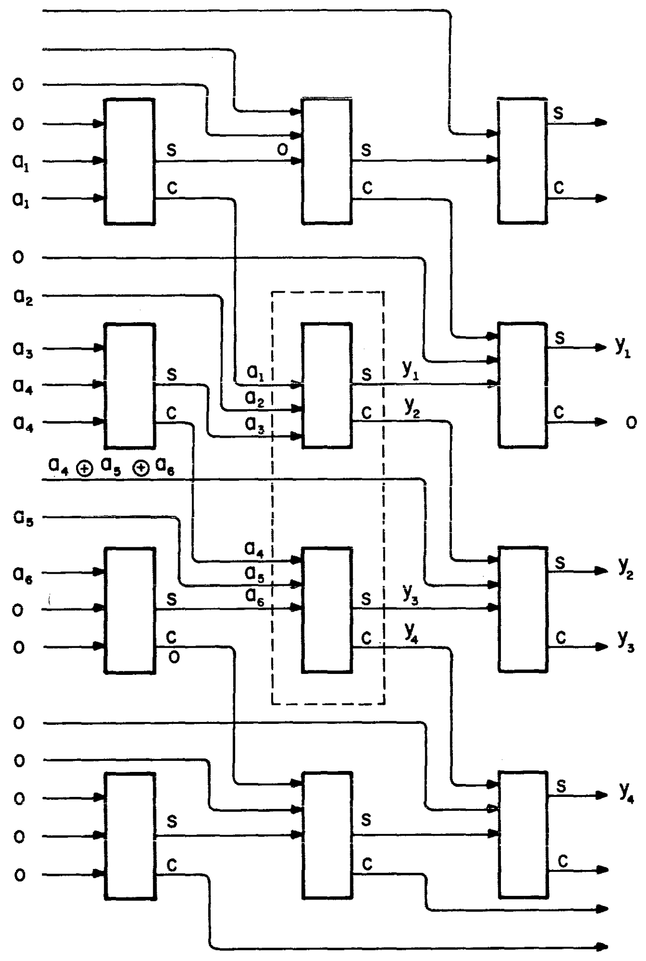


Figure 7—Solution to Figure 5

Location of errors

The determination of nodes and arcs relevant to each path test is straightforward from the path matrix. Translation from nodes and arcs to packages and wiring then enables us to determine which physical components of the machine affect a test result.

Location of single errors detected in path tests is possible by comparing the columns of the path matrix with each test result expressed in a vector form in which a 1 indicates failure and a 0 success of the test. In effect, simultaneous comparison of many columns with a test result can be done by taking a logical product of the matrix rows complemented if the result of the test corresponding to the row is a success.

Multiple errors can be located, though in a less straightforward way and with a lower resolution, by selecting a maximal set of columns each of which implies the test result vector.

Another method for locating errors consists of organizing a branching tree of tests so that each terminal of the tree corresponds to a distinguishable error or a set of indistinguishable errors. The execution of tests follows a chain of tests in the tree selecting either a "success" branch or a "failure" branch after each test case until the termination.[8]

The following theorem is evident from Theorem 1 on the minimality of path set generated by the PGM.

Theorem 2: When a set of test paths generated by the PGM is developed into a branching tree, the number of tests in the tree (which is the number of tree nodes excluding terminals) is the same regardless of which paths are assigned to the tests, provided that there are no redundant tests.

By a redundant test is meant a test of a path which passes through all nodes and arcs remaining to be distinguished at the test point or which passes through none of them.

Therefore, location of single errors by a branching tree of path tests is an even more straightforward procedure than the comparison of matrix columns. Also, multiple errors (although perhaps not all of them) can be located by an organization of test tree based on the evaluation of a proper weight for each test.

Errors detected in the combinational tests may not be located in an equally straightforward manner; however, their locations are identified at least up to a logic block and a further isolation can be done by manipulation of a matrix similar to the path matrix.

While the equation (5) guarantees that any error can be observed at some of the external outputs, a Boolean difference dz_k/dy_j specifies the condition on the external inputs under which any error at signal y_j is detected at the specified external output z_k. Therefore the matrix can be obtained by evaluating the difference for each pair of y_j and z_k.

When a matrix manipulation is not possible or realistic enough in practice, a test dictionary is produced using the methods above and the maintenance engineers can look up the most suspicious error positions listed in it during the actual testing.

Logic simulation of the machine

It is a common practice to use a logic simulation program to determine the correct response of the machine to each test input.

A logic simulator may also be used extensively in debugging the logic design. Because diagnostic generation may proceed in parallel with logic debugging (without waiting for its completion), the construction of the simulator should be as exact as possible to the mechanical organization of the logic. For discrete circuitry computer systems, the fastest simulators used to be compiled from a gate-level description of the design, a gate in the machine corresponding to a few machine instructions (Boolean or where not available, arithmetic operations) in the simulation program. The ordering of the instructions was a well-defined ordering among the gates. The introduction of integrated circuitry and higher integration into the computer, however, brought in some new features in logic design and simulation.

Logic design cannot always precede the package design. Physical and mechanical limitations on semiconductor and packaging technology place significant restrictions on logic design and it is not possible for a machine to be designed completely in detail and then partitioned. The logic of the machine must be described in at least two levels—the logic of every package type and the interpackage connections. Thus it is less straightforward than it used to be to compile a gate-level simulator.

For the same reason any change in design may be reflected in different ways in the simulator. Either a logic package is modified, or a connection is replaced by another, or both. A simulator can be much more easily updated if the logic packages and connections can be distinguished one from the other.

On the other hand, the turn-around time in designing and manufacturing high density packages cannot be neglected. The logic internal to a package may not be changed as easily as the connections among packages. The construction of the logic simulator may have the same nature.

Because of these factors subroutines can be utilized to describe the logic package. When a simulator is created using a general-purpose programming language (ALGOL, FORTRAN, or PL/1), a procedure can define the relations between the inputs and outputs of a package. Then the body of the simulator becomes a sequence of procedure call statements, the actual parameters associated with each call corresponding to the signals incident to the package.

The procedure calls must be ordered properly so that the propagation of control and data signals can be followed. There are two problems arising in the ordering of packages. They are:

1. To which level to assign packages containing storage elements.
2. How to order the packages involved in loops.

The first problem can be solved by duplicating a register package into two nodes—corresponding to the loading and output selection functions of the package—in the graph representation. Thus, the receiver and register-output nodes are assigned the first levels, while the drivers and register-input nodes are assigned the last.

The packages involved in a loop (or a maximal "circuit," or a "maximal connected subgraph") are assigned the same level. The associated procedure calls are put in a program loop and executed until the values of inter-package signals in the loop are all stabilized.

Such an assignment can be efficiently programmed in the following steps:

1. Assign levels from the first level in an ascending order and then from the last in a descending order until the remaining nodes cannot be assigned any unique level.
2. Using matrix operations on the adjacency matrix, find all the maximal loops and reduce each of them to a dummy node.
3. Assign the remaining levels using the newly developed dummy nodes in place of loops.

CONCLUSIONS

A method of generating test cases for diagnosing a machine using high density circuitry has been described. This method is motivated by the assumption that computer organizations are becoming more modular and the failure modes of high density integrated circuits are becoming more complicated and less obvious.

A minimal and sufficient set of path tests can be generated by a simple algorithm working on a graph representation of the machine.

Combinational logic networks can be diagnosed by a set of test input patterns generated by a procedure described. There are three sets of equations which must be simultaneously solved to obtain input patterns; one of them is a new requirement caused by the high density of circuits in a semiconductor chip.

A logic simulator may also reflect the progress of hardware technology. A low-cost simulator generator has been developed which uses a general purpose programming language as the simulator media, in which the package logic is described as functional procedures.

Several methods of fault location have also been described for path tests and combinational tests.

The Path Generating Method has been programmed in Extended ALGOL on Burroughs B5500. Generation of combinational tests were performed semiautomatically. A logic simulator generator has also been programmed on B5500. A logic simulator of the Processing Element (PE) of the ILLIAC IV Computer was generated by the program and has been used successfully in logic debugging and diagnostic generation.

Path and combinational tests generated by the described methods have been employed in off-line testing of the PE. On-line testing will use another set of tests produced by the same methods.

ACKNOWLEDGMENT

The authors wish to express their gratitude to Professor D. L. Slotnick and certain members of the staff in the ILLIAC IV project for their advice and discussions.

The algorithm of the Path Generating Method was developed through discussions with Messrs. W. L. Heimerdinger and Y. Matsushita of the University of Illinois; programming efforts were done by Messrs. C. Chen, J. E. Miller, W. Sterling, and C. Tanaka.

This work was supported in part by the Department of Computer Science, University of Illinois, Urbana, Illinois, and in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center, under Contract No. USAF 30(602)4144.

REFERENCES

1 J SESHU D N FREEMAN
The diagnosis of asynchronous sequential switching systems
IRE Trans on Electronic Computers August 1962

2 S HASHIMOTO T KASAMI H OZAKI
Fault diagnosis of combinational logical networks
Journal of Inst of Electrical Communication Engineers Japan April 1964

3 D B ARMSTRONG
On finding a nearly minimal set of fault detection tests for combinational logic nets
IEEE Trans on Electronic Computers February 1966

4 W H KAUTZ
Fault testing and diagnosis in combinational digital circuits
IEEE Trans on Computers April 1968

5 J P ROTH W G BOURICIUS P R SCHNEIDER
Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits
IEEE Trans on Electronic Computers October 1967

6 C V RAMAMOORTHY
A structural theory of machine diagnosis
Proc S J C C 1967

7 F F SELLERS JR M Y HSIAO L W BEARSON
Analyzing errors with the Boolean difference
IEEE Trans on Computers July 1968

8 H Y CHANG
An algorithm for selecting an optimum set of diagnostic tests
IEEE Trans on Electronic Computers October 1965

9 G H BARNES R M BROWN M KATO
J D KUCK D L SLOTNICK R A STOKES
The ILLIAC IV computer
IEEE Trans on Computers August 1968

Programmed test patterns for multiterminal devices

by FRANCIS J. McINTOSH and W. W. HAPP

National Aeronautics and Space Administration
Cambridge, Massachusetts

INTRODUCTION

The rapid development of micro-electronics towards multiterminal structures demands corresponding growth in understanding the potential and limitations of multiterminal devices and networks. The increasing sophistication of integrated circuits will impose a new set of criteria upon network synthesis.

In particular, through the suitable arrangements of terminals or *accessible test points* of a multiterminal device, many distinct configurations may be realized. A multiterminal network like the transistor contains several realizable configurations, typically a voltage amplifier, a current amplifier, an attenuator, or a filter. Hand enumeration will suffice in attaining all of the configurations derivable from such a small network. However, several thousand distinct functions can be realized from a six-terminal network, and the scope of network synthesis could well be directed to selecting one of several million functions available from a multiterminal network. A computer program has been developed to enumerate all of the allowable configurations so that each may be identified uniquely and non-redundantly.

Scope of investigation

Review of related work

Previous investigations of multiterminal devices utilized combinatorial techniques to obtain algorithms for generation of network functions and failure diagnostics:

a. the number N(z, x, y) of x-terminal y-port subnetworks derivable from a z-terminal parent network was tabulated.[1]
b. the separation of variables as a product of functions is possible so that

$$N(z, x, y) = H(z, x) \cdot T(y, x - y),$$

where H(z, x) is related to the partitioning of z, and T(y, x − y) is related to the number of trees of a structure with x components.[2]
c. a computer program implements the algorithms contained in the previous papers to calculate the numbers H, T, and N, but does not list these functions.[3]

An extensive bibliography is provided in the references cited.

Objectives

The objective of this investigation is the generation and listing of the H(z, x) subnetwork configurations, which result when a z-terminal parent network is constrained to form x-terminal subnetworks. The computer program developed will generate and list explicitly all unique and non-redundant x-terminal subnetwork configurations of a z-terminal parent network. The assumptions under which the program is formulated are:

a. the z-terminal network or device asymmetrical;
b. the z parent terminals may be utilized to form terminals of the subnetwork, or may be left free or uninvolved in the subnetwork configurations;
c. subnetwork configurations are arrived at through the systematic shorting and opening of accessible terminals.

Based on this work future investigations are planned:

a. to develop techniques for dealing with the symmetry of multiterminal devices and obtain a non-redundant listing of the H(z, x) configurations with specified symmetry constraints;

b. to provide a computer program for listing the T(y, x-y) configurations, and subsequently obtain an explicit list of the N(z, x, y) configurations with and without symmetry constraints;
c. to apply these listings to failure diagnostics of multiterminal structures;
d. to select network performance characteristics on the basis of subnetwork listings.

Illustrative example

To illustrate the concepts to be used and the terminology to be defined subsequently more rigorously, the configuration H(4, 2) is examined in Figure 1. Six subnetwork configurations, illustrated on the left, are realizable from a device with z = 3 and x = 2 by utilizing either:

a. two of the three terminals of the parent network and leaving the third terminal disconnected, or
b. all three externally connecting two together to form one terminal.

Larger networks form thousands of subnetwork configurations and hence require a systematic coding procedure. One method is illustrated in Figure 2 and Figure 3.

a. the z parent terminals are arbitrarily numbered;
b. each line (or word) represents a subnetwork in code;
c. numerals refer to external terminals of the subnetwork;

```
 1*12AA
 2*12AB
 3*1A2A
 4*1A2B
 5*1AA2
 6*1AB2
 7*A12A
 8*A12B
 9*A1A2
10*A1B2
11*AA12
12*AB12

13*112A
14*121A
15*211A
16*11A2
17*12A1
18*21A1
19*1A12
20*1A21
21*2A11
22*A112
23*A121
24*A211

25*1112
26*1121
27*1211
28*2111

29*2211
30*2121
31*2112
```

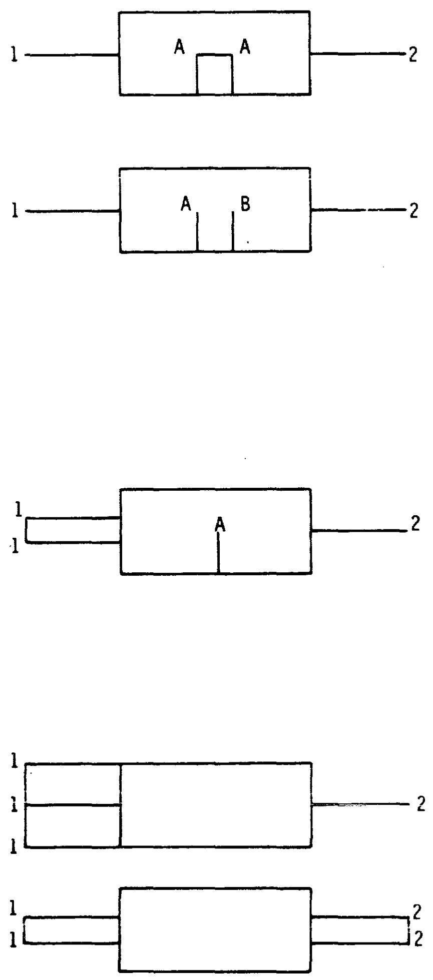


Figure 1—Print-out of H(4,2) configurations

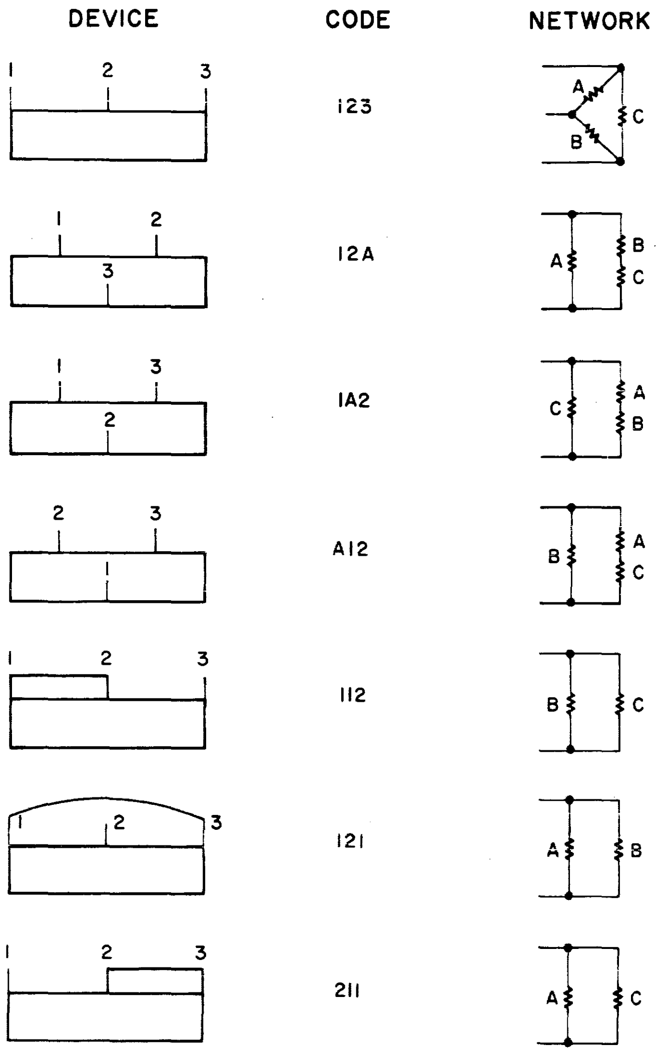


Figure 2—Code for H(3,2) configurations

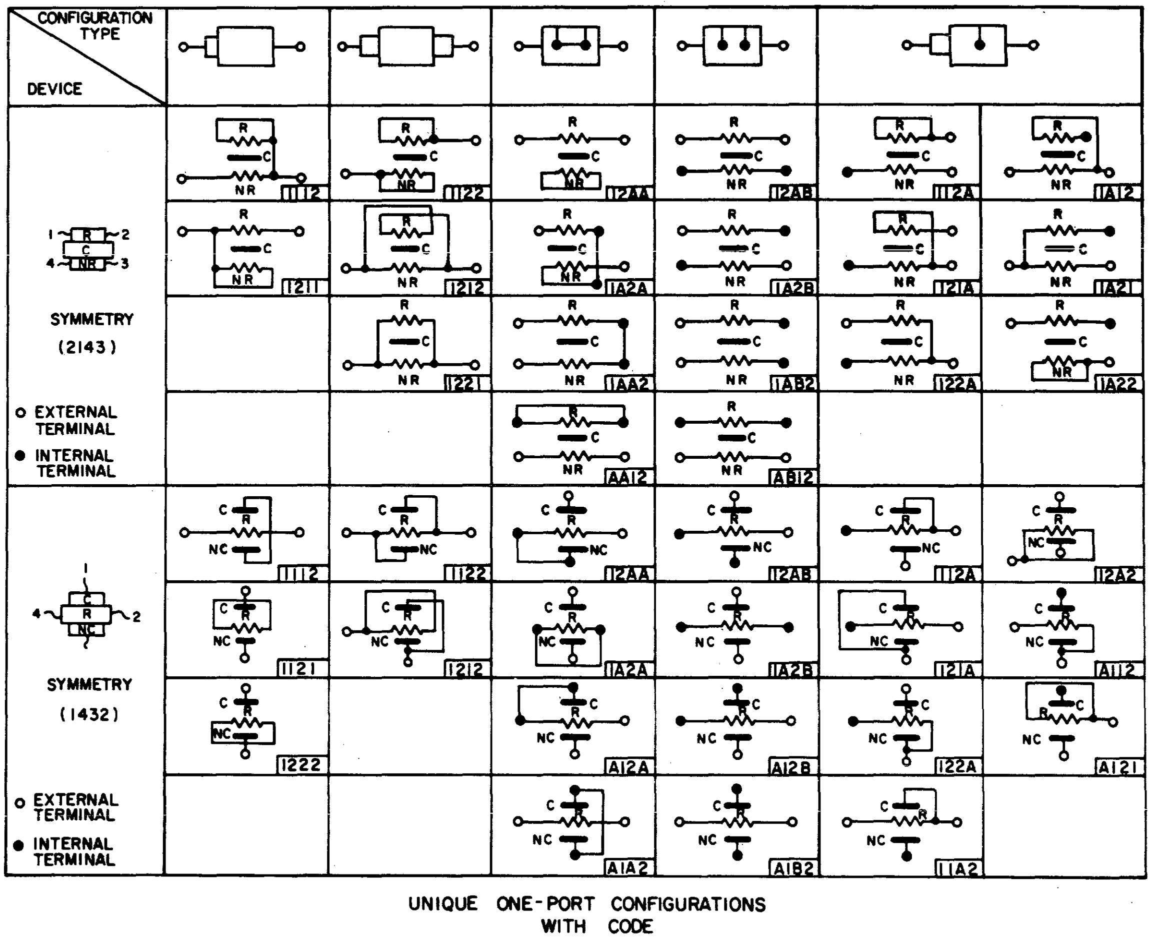


Figure 3—Unique one-port configurations with code

d. letters refer to the internal terminals, which are not part of the subnetwork;
e. position of number or letter in a word indicates the terminal of the parent network represented by the number or letter.

The coded computer print-out may then be applied to a particular network device. The H(3, 2) configurations are illustrated on the right in Figure 1. Given the computer output for H(3, 2) and the parent network structure, each of the six subnetwork circuit configurations is readily identifiable.

Algorithms for generating subnetworks

Terminology

- A *terminal* is an accessible testpoint in the parent network, each terminal will become either an external or internal terminal in the subnetwork.
- A *multiterminal structure* is a network or device of more than two terminals. The computer program is developed for structures of up to eight terminals.
- An *external terminal* of the subnetwork is an accessible testpoint consisting of either a single terminal of the parent network *or* a group of terminals from the parent network constrained to form the single external terminal.
- An *internal terminal* in the subnetwork configuration is a parent terminal(s) which is (are) left free or uninvolved in the subnetwork and thereby constitute unaccessible testpoints in the subnetwork configurations.
- An *external configuration* is a unique and non-redundant arrangement of one or more terminals of the parent network at each terminal of the subnetwork.
- An *internal configuration* is unique and non-redundant arrangements of terminals not forming part of the subnetwork in groups of one or more. In diagrams the internal configurations are drawn inside the system to differentiate them from external configurations.

Separation into external and internal configurations

The program is based on the following procedure:

a. the *external configurations* are defined by the restriction of I terminals, $x \leq I \leq z$, to form x external terminals;
b. the z-I terminals remaining are constrained into groups of 1, 2, . . . , z-I to form the *internal configurations*, each group representing a unique and non-redundant configuration;
c. varying I in unit steps from x to z, *enumerate and store* the external and internal terminal configurations for each I and z-I respectively;
d. integrate by appropriate combinatorial techniques the stored external and internal configurations at each step to yield the desired H(z, x) configurations.

To accomplish this, eight subprogram subroutines are utilized in the program, three of which—EXTERN, INTERN, INTEG—are called in succession by a small master program, and which in turn utilize the other five in appropriate sequences according to the flowchart.[3] Two techniques are of fundamental importance to these subroutines:

a. Enumeration of Combinations in Binary Code
b. Disjoint Loop Enumeration and Storage

These techniques will be discussed now.

Enumeration of combinations in binary code

The enumeration technique is based upon the subroutines BIN and MAIN which generate combinations of P things taken Q at a time. Each combination is generated as a series of ones and zeros, yielding Q "ones" and P-Q "zeros". The terminal numbers under which the ones fall defined terminals included in each particular combination with P digit-positions in all. By systematically moving the Q ones into all combinations of the P positions, the resultant (P, Q) combinations are explicitly listed and yield the number which is the binomial coefficient of P and Q.

Each combination is then stored in a so called "compendious form", a technique previously used in processing of combinatorial information.[4,5] Treating each combination of ones and zeros as a number of base 2, the word is changed to decimal notation as, for example, for $P = 4$ and $Q = 2$ in Figure 4. This is accomplished by treating each combination as a series x_i $i = 1, \ldots, P$, and performing the operation $F(x) = \sum_{i=1}^{P} x_i \cdot 2^{i-1}$.

$$1100 = 1+2 = 3$$
$$1010 = 1+4 = 5$$
$$1001 = 1+8 = 9$$
$$0110 = 2+4 = 6$$
$$0101 = 2+8 = 10$$
$$0011 = 4+8 = 12$$

Figure 4—Transformation of $P = 4$, $Q = 2$ into decimal notation

Each decimal number thus created is saved as a single subscripted integer variable in BIN and as a double subscripted integer variable in MAIN.

This method of storage is advantageous for two reasons. The computer memory utilizes binary coding in its storage of a decimal number: each combination thus coded may be recalled from its decimal number. Efficiency and economy also results by utilizing only one storage position in memory per combination.

Disjoint loop enumeration and storage

The routine which enumerates and stores disjoint loops exploits an isomorphism between the allocation of external and internal terminals for a subnetwork and the identification of disjoint loops in a flowgraph. If each loop of a flowgraph is coded in binary form in terms of the nodes it contains, then disjoint loops can be identified by binary addition.[6,7,8] The resultant subroutines calculate and store those combinations of loops which have no nodes in common.

The logical operation .AND. compares two binary-coded decimal numbers. If two loops are disjoint the resultant combination is defined as a second-order loop. The number of disjoint combinations involved in the union is defined as the loop order. Thus, a loop of order m consists of m first order loops which have no nodes in common.

The loops are stored as a subscripted integer variable which is made up of the subscripts of the integer variables under which the disjoint binary-coded numbers are stored. Combinations of terminals correspond to loops or combinations of nodes, each node corresponding to a terminal of the parent network. In the previous example of (4, 2) the loops 12 and 3 are disjoint. Labeling first order loops sequentially, consider row six and row one, a second order loop results which are stored as 601. Similarly, 11 and 5, 6 and 9 are disjoint binary-coded numbers, and are stored under their subscripts as 502 and 403 respectively. In this way the binary-coded numbers and the combinations which they represent are stored in a compendious manner and may be efficiently recalled.

Sequence of subroutines

The generation H(z, x) for a particular I external terminals and z-I internal terminals is described by comment cards defining the function performed by each subroutine and is given in a listing or the program.[3] A few additional comments appear in order.

Internal configurations

After the problem statement provides the program with z and x, INTERN is called for IQ = z − I terminals. This subroutine in turn calls BIN to enumerate and save the combinations of IQ terminals taken 2, 3, . . . , IQ at a time in the stored binary code. The combinations of IQ items taken one at a time are taken into effect and enumerated later in INTERN. The disjoint loops of the above are next calcualted by INTERN and stored. SPRINT is utilized to break down the stored disjoint loops into their unique configurations and enumerate in numeric code the proper internal configurations implicitly represented by each stored loop. These configurations form the matrix IITER (I, J).

External configurations

The suroutine EXTERN is next referenced for I and x terminals. The partition numbers subroutine PART is then called by EXTERN to return how I terminals are restrained to yield x terminals.
For example:

$$P\ (4, 2) = \begin{bmatrix} 2, 2 \\ 3, 1 \end{bmatrix}$$

For each number in a row of partition numbers, the combinations of I terminals taken that number at a time are enumerated and stored.

Once the combinatorial enumeration of a particular row is accomplished, subroutine LPORD is utilized to calculate the disjoint loops for those combinations of the row.

Since redundant loops may be generated for a particular row if two or more numbers in that row are equal, this case is accounted for and only non-redundant loops are enumerated. As each loop is calculated the proper numeric code is generated for the external configuration of which it is representative. By repeating the process for each row of partition numbers, the matrix of external configurations IETER (I, J) is formed.

Integration of internal and external configurations

The final subprogram subroutine INTEG is called as soon as the matrices IITER (I, J) and IETER (I, J) are complete for a particular IQ and I. These matrices are then integrated systematically as determined by the ways of combining z terminals I at a time, and as enumerated by BIN in ones and zeros. The number of ones equals I and the number of zeros equals IQ. For a particular I external terminals and z − I = IQ internal terminals, the H(z, x) configurations are listed through the substitution of entries of the IETER (I, J) matrix and IITER (I, J) matrix for the ones and zeros, respectively, of each combination enumerated by BIN.

Output format for configurations

The configurations thus generated must be represented in a unique and convenient code. Previous work[1,2] utilized upper case letters for external terminals and lower case letters for internal terminals, identical letters and type indicating those terminals of the parent network that are joined to form the subnetwork configuration. Since a computer will only print upper case letters, the lower case letters were replaced by numeric character. A variation of this alphanumeric code is used up to a point just prior to print-out.

The pre-print-out format for each configuration is a series of positive and negative numeric characters. Positive integers correspond to external subnetwork terminals; negative integers correspond to internal subnetwork terminals. Identical integers and signs indicate terminals of the parent network which are joined to form the subnetwork configuration. This coding scheme is perhaps the most applicable to a computer setup for testing multiterminal devices and for network analysis. An adaptation of this format is the substitution of upper case alphabetic characters for the negative integers, since this scheme provides for greater visual clarity in identifying the unique configurations. A listing of a number of subnetwork configurations is found in Figures 5 and 6.

Applications

Identification of unique network functions

The computer print-out for H(4, 2) is given in Figure 1. These coded subnetworks are relevant to testing and

```
H( 3 2)

1*12A
2*1A2
3*A12
4*112
5*121
6*211

H( 4 2)

1*12AA
2*12AB
3*1A2A
4*1A2B
5*1AA2
6*1AB2
7*A12A
8*A12B
9*A1A2
10*A1B2
11*AA12
12*AB12
13*112A
14*121A
15*211A
16*11A2
17*12A1
18*21A1
19*1A12
20*1A21
21*2A11
22*A112
23*A121
24*A211
25*1112
26*1121
27*1211
28*2111
29*2211
30*2121
31*2112

H( 5 2)

1*12AAB
2*12ABA
3*12BAA
4*12AAA
5*12ABC
6*1A2AB
7*1A2BA
8*1B2AA
9*1A2AA
10*1A2BC
11*1AA2B
12*1AB2A
13*1BA2A
14*1AA2A
15*1AB2C
16*1AAB2
17*1ABA2
18*1BAA2
19*1AAA2
20*1ABC2
21*A12AB
22*A12BA
23*B12AA
24*A12AA
25*A12BC
26*A1A2B
27*A1B2A
28*B1A2A
29*A1A2A
30*A1B2C
31*A1AB2
32*A1BA2
33*B1AA2
34*A1AA2
35*A1BC2
36*AA12B
37*AB12A
38*BA12A
39*AA12A
40*AB12C
41*AA1B2
42*AB1A2
43*BA1A2
44*AA1A2
45*AB1C2
46*AAB12
47*ABA12
48*BAA12
49*AAA12
50*ABC12
51*112AA
52*121AA
53*211AA
54*112AB
55*121AB
56*211AB
57*11A2A
58*12A1A
59*21A1A
60*11A2B
61*12A1B
62*21A1B
63*11AA2
64*12AA1
65*21AA1
66*11AB2
67*12AB1
68*21AB1
69*1A12A
70*1A21A
71*2A11A
72*1A12B
73*1A21B
74*2A11B
75*1A1A2
76*1A2A1
77*2A1A1
78*1A1B2
79*1A2B1
80*2A1B1
81*1AA12
82*1AA21
83*2AA11
84*1AB12
85*1AB21
86*2AB11
87*A112A
88*A121A
89*A211A
90*A112B
91*A121B
92*A211B
93*A11A2
94*A12A1
95*A21A1
96*A11B2
97*A12B1
98*A21B1
99*A1A12
100*A1A21
101*A2A11
102*A1B12
103*A1B21
104*A2B11
105*AA112
106*AA121
107*AA211
108*AB112
109*AB121
110*AB211
111*1112A
112*1121A
113*1211A
114*2111A
115*2211A
116*2121A
117*2112A
118*111A2
119*112A1
120*121A1
121*211A1
122*221A1
123*212A1
124*211A2
125*11A12
126*11A21
127*12A11
128*21A11
129*22A11
130*21A21
131*21A12
132*1A112
133*1A121
134*1A211
135*2A111
136*2A211
137*2A121
138*2A112
139*A1112
140*A1121
141*A1211
142*A2111
143*A2211
144*A2121
145*A2112
146*11112
147*11121
148*11211
149*12111
150*21111
151*22211
152*22121
153*22112
154*21221
155*21212
156*21122
157*12221
158*12212
159*12122
160*11222

H( 4 3)
1*123A
2*12A3
3*1A23
4*A123
5*1123
6*1213
7*1231
8*2113
9*2131
10*2311

H( 5 3)
1*123AA
2*123AB
3*12A3A
4*12A3B
5*12AA3
6*12AB3
7*1A23A
8*1A23B
9*1A2A3
10*1A2B3
11*1AA23
12*1AB23
13*A123A
14*A123B
15*A12A3
16*A12B3
17*A1A23
18*A1B23
19*AA123
20*AB123
21*1123A
22*1213A
23*1231A
24*2113A
25*2131A
26*2311A
27*112A3
28*121A3
29*123A1
30*211A3
31*213A1
32*231A1
33*11A23
34*12A13
35*12A31
36*21A13
37*21A31
38*23A11
39*1A123
40*1A213
41*1A231
42*2A113
43*2A131
44*2A311
45*A1123
46*A1213
47*A1231
48*A2113
49*A2131
50*A2311
51*11123
52*11213
53*11231
54*12113
55*12131
56*12311
57*21113
58*21131
59*21311
60*23111
61*22113
62*22131
63*22311
64*21213
65*21231
66*23211
67*21123
68*21321
69*23121
70*21132
71*21312
72*23112
73*32211
74*32121
75*32112

H( 5 4)

1*1234A
2*123A4
3*12A34
4*1A234
5*A1234
6*11234
7*12134
8*12314
9*12341
10*21134
11*21314
12*21341
13*23114
14*23141
15*23411

H( 6 4)

1*1234AA
2*1234AB
3*123A4A
4*123A4B
5*123AA4
6*123AB4
7*12A34A
8*12A34B
9*12A3A4
10*12A3B4
11*12AA34
12*12AB34
13*1A234A
14*1A234B
15*1A23A4
16*1A23B4
17*1A2A34
18*1A2B34
19*1AA234
20*1AB234
21*A1234A
22*A1234B
23*A123A4
24*A123B4
25*A12A34
26*A12B34
27*A1A234
28*A1B234
29*AA1234
30*AB1234
31*11234A
32*12134A
33*12314A
34*12341A
35*21134A
36*21314A
37*21341A
38*23114A
39*23141A
40*23411A
41*1123A4
42*1213A4
43*1231A4
44*1234A1
45*2113A4
46*2131A4
47*2134A1
48*2311A4
49*2314A1
50*2341A1
51*112A34
52*121A34
53*123A14
54*123A41
55*211A34
56*213A14
57*213A41
58*231A14
59*231A41
60*234A11
61*11A234
62*12A134
63*12A314
64*12A341
65*21A134
66*21A314
67*21A341
68*23A114
69*23A141
70*23A411
71*1A1234
72*1A2134
73*1A2314
74*1A2341
75*2A1134
76*2A1314
77*2A1341
78*2A3114
79*2A3141
80*2A3411
81*A11234
82*A12134
83*A12314
84*A12341
85*A21134
86*A21314
87*A21341
88*A23114
89*A23141
90*A23411
91*111234
92*112134
93*112314
94*112341
95*121134
96*121314
97*121341
98*123114
99*123141
100*123411
101*211134
102*211314
103*211341
104*213114
105*213141
106*213411
107*231114
108*231141
109*231411
110*234111
111*221134
112*221314
113*221341
114*223114
115*223141
116*223411
117*212134
118*212314
119*212341
120*232114
121*232141
122*232411
123*211234
124*213214
125*213241
126*231214
127*231241
128*234211
129*211324
130*213124
131*213421
132*231124
133*231421
134*234121
135*211342
136*213142
137*213412
138*231142
139*231412
140*234112
141*322114
142*322141
143*322411
144*321214
145*321241
146*324211
147*321124
148*321421
149*324121
150*321142
151*321412
152*324112
153*342211
154*342121
155*342112
```

Figure 5—Short reference table for test patterns

network analysis since, as two-terminal configurations, they identify unique driving-point functions. The technique of generating the number of external terminals is illustrated in Figure 3. Configurations can be identified which have the same external configurations, but differ in that their internal terminals are joined or left disconnected.

From H(4, 2) in Figure 1 it is feasible to generate network functions from four-ports of known symmetry. Two examples, Figure 7, define the problem to be solved next; namely, to eliminate redundance due to symmetry of the network.

Thin-film RC networks

Thirteen unique network functions are attained from both device A and B in Figure 7. Each circuit configuration is represented by one of the thirty-one H(4, 2) configurations, and may be identified by the letter A or B next to the appropriate coded H(4, 2) word and under the circuit it identifies. The eighteen coded words not representative of unique network functions, do identify redundant configurations due to rotation of the device about its axis of symmetry. Redundant configurations are designated by $\bar{A}$ and $\bar{B}$ under the appropriate circuit configurations.

Reference tables for test patterns

To test devices and systems with a small number of terminals, say $z<10$, reference tables to identify unique test patterns are desirable. From the test pat-

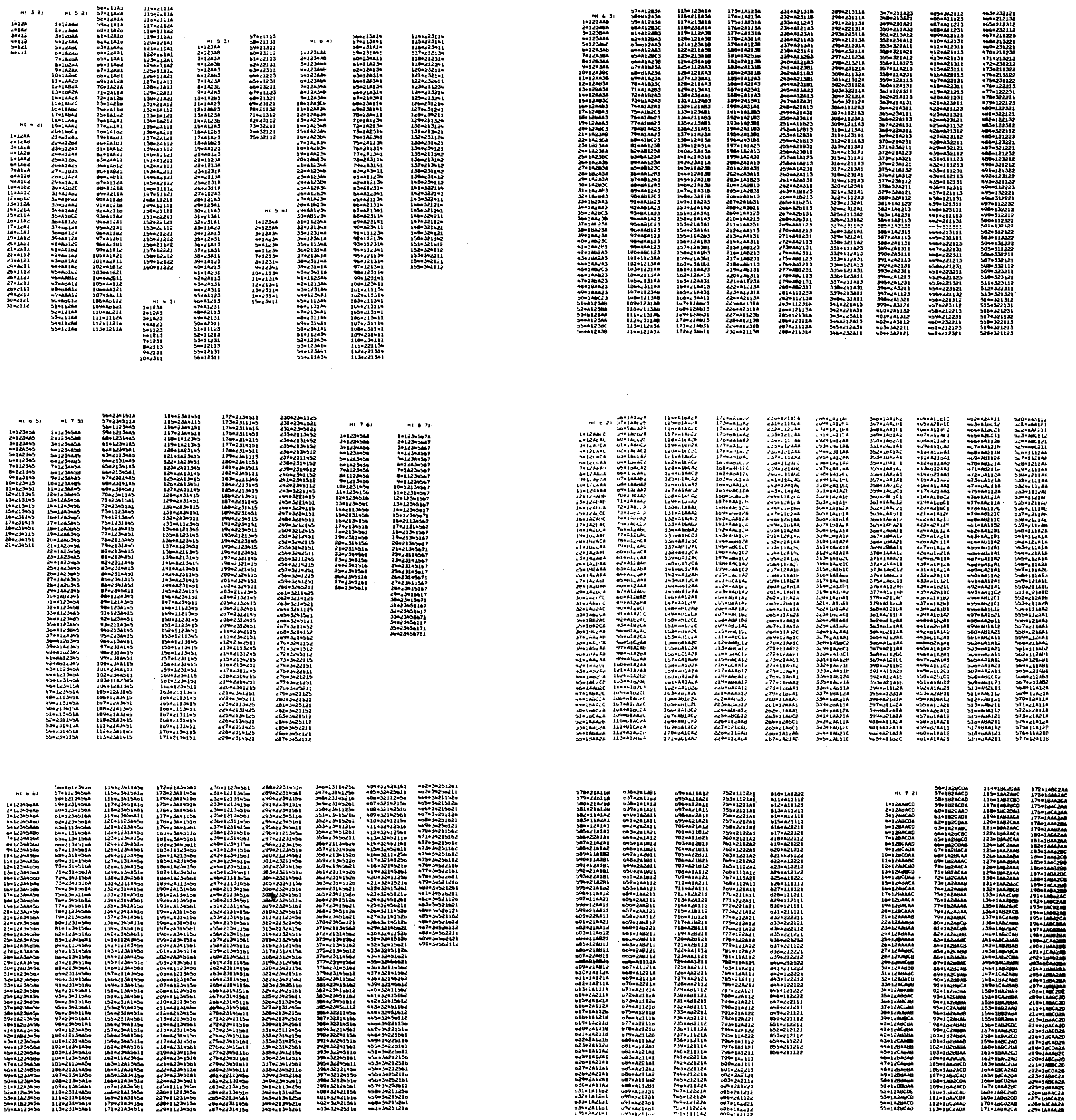
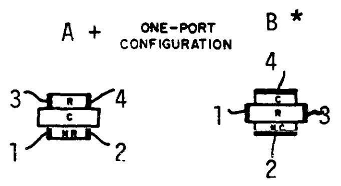
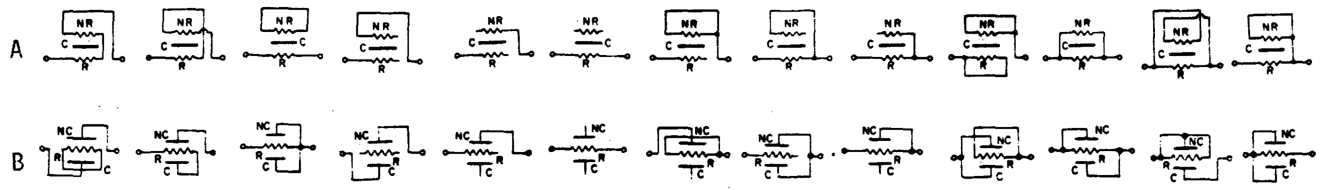


Figure 6—Sample of reference table for test patterns

terns given in Figure 6, optimum test sequences can be constructed for use in testing LSI packages and other microelectronic structures. A further use of these tables occurs in failure identification techniques based on a systematic scan of all configurations. It is anticipated that these tables will fulfill a similar purpose as the table of partition number; namely, usefulness for paper and pencil calculations which do not justify utilization of computer programs. For applications involving a greater number of test prints, computer programs are essential. The total number of test patterns shown in Figure 8 listing H(z, x) for $z \leq 12$, ex-





12AA		B											
12AB					B								
1A2A	A		B										
1A2B				A		B							
1AA2													
1AB2													
A12A													
A12B					A								
A1A2	B	A											
A1B2				B									
AA12			A										
AB12						A							
112A							A						
121A							B						
211A									B				
11A2													
12A1													
21A1								B					
1A12													
1A21								A					
2A11													
A112													
A121									A				
A211													
1112												B	
1121													AB
1211													
2111													
2211										A	B		
2121										B	A		
2112												A	

Figure 7—Unique subnetworks for device with specified symmetry

z/x	1	2	3	4	5	6	7	8	9	10	11	12
1	1											
2	3	1										
3	10	6	1									
4	37	31	10	1								
5	151	160	75	15	1							
6	674	856	520	155	21	1						
7	3263	4802	3556	1400	287	28	1					
8	17007	28337	24626	11991	3290	490	36	1				
9	94828	175896	174805	101031	34671	6972	786	45	1			
10	562595	1146931	1279240	853315	350889	88977	13620	1200	55	1		
11	3535027	7841108	9677151	7300260	3492511	1069068	207537	24915	1760	66	1	
12	23430840	56089804	75750752	63641006	34669734	12428746	2928684	447612	43175	2497	78	1

Figure 8—Listing of H(z,x) unique subnetworks

tends previously published results.[1]

Future investigations

Problems to be solved

The computer program presented here represents a step in the explicit formulation of the N(z, x, y) subnetwork configurations. For optimum exploitations of the advantages of this program two additions must be made:

a. A routine must be written to take into effect the symmetry of devices and networks.
b. A separate computer program must be added to generate and list the T(y, x-y) configurations, and thus obtain a complete enumeration of N(z, x-y) as defined in an earlier section.

Methods under investigation

The symmetry problem may be approached through the use of a tagging procedure which will identify the axes of symmetry of the device, as well as any peculiarity of the device which must be considered.[9]

Figure 9 is a print-out for the H(6, 2) configuration, asterisks denote successive elimination;
(*) by symmetry (543216)
(**) by symmetry (216543) and
(***) by combination of both symmetries (426123).

Proposed strategy

Upon solution of the two problems discussed above, the computer program will be developed. The proposed strategies are:

a. Enlarge the program to accept devices and networks of more than eight terminals. This can be readily accomplished since much of the program is written in modular form.
b. Test the adequacy of the program in a laboratory situation by applying subnetwork listings to failure diagnostics of multiterminal structures and to selecting network performance characteristics on the basis of subnetwork listings. Pin-to-pin testing has already been effectively used for screening of multiterminal integrated circuits.[10,11]
c. Compare combinatorial techniques with alternative computer-oriented approaches for fault isolation, such as performance variation analysis and worse case design.[12,13] A detailed critical assessment of computer-aided fault identification techniques is given in the current issues of the Reliability Abstracts.[14]

Figure 10 defines unique two-ports for two symmetrical devices, as yet algorithms to define corresponding test patterns are not available.

CONCLUSION

A computer program, primarily relevant to microelectronics, has been developed to generate and list the many distinct configurations of a multi-terminal network or device, realizable through the suitable arrangement of the terminals or accessible test points. The program is unique in that it utilizes combinatorial techniques to generate all of the non-redundant subnetwork configurations derivable from an asymmetrical network or device. This is accomplished by a systematic shorting and opening of accessible terminals to obtain the desired allowable configurations.

The prime advantage of the program lies in the fact that for large networks of seven or eight terminals, thousands of unique configurations can be realized. Hand enumeration will not suffice in attaining all of these configurations.

The program is applicable to testing and network analysis. In particular, the computer print-out provides test programs for devices and networks with a small number of terminals. Failure identification procedures based on a systematic scan of all configurations can utilize such test programs.

REFERENCES

1 W W HAPP
Combinatorial analysis of multi-terminal device
IEEE transactions on systems science and cybernetics
Vol. SSC-3 No. 1 June 1967 21-27

2 A S WEITZENFELD W W HAPP
Combinatorial algorithms for computer-oriented fault identification in multiterminal devices
Proceedings IEEE-ESD Symposium on Maintainability
Concord Mass April 1967 Also published in revised form as
Combinatorial techniques for fault identification in multiterminal networks
IEEE Transactions on Reliability Vol R-16 No. 3

```
1*12AABC
2*12ABAC
3*12ABCA
4*12ABBC
5*12ABCB **
6*12ABCC **
7*12AAAB
8*12AABA
9*12ABAA **
10*12ABBB **
11*12AAAA
12*12AABB
13*12ABAB
14*12ABBA
15*12ABCD
16*1A2ABC
17*1A2BAC
18*1A2BCA
19*1A2BBC
20*1A2BCB
21*1A2BCC
22*1A2AAB
23*1A2ABA
24*1A2BAA
25*1A2BBB
26*1A2AAA
27*1A2ABB
28*1A2BAB
29*1A2BBA
30*1A2BCD
31*1AA2BC
32*1AB2AC
33*1AB2CA
34*1AB2BC ***
35*1AB2CB
36*1AB2CC ***
37*1AA2AB
38*1AA2BA
39*1AB2AA ***
40*1AB2BB ***
41*1AA2AA
42*1AA2BB
43*1AB2AB
44*1AB2BA
45*1AB2CD
46*1AAB2C
47*1ABA2C
48*1ABC2A
49*1ABB2C *
50*1ABC2B

51*1ABC2C *
52*1AAA2B
53*1AAB2A
54*1ABA2A
55*1ABB2B *
56*1AAA2A
57*1AAB2B
58*1ABA2B
59*1ABB2A *
60*1ABC2D
61*1AABC2
62*1ABAC2
63*1ABCA2
64*1ABBC2
65*1ABCB2
66*1ABCC2
67*1AAAB2
68*1AABA2
69*1ABAA2
70*1ABBB2
71*1AAAA2
72*1AABB2
73*1ABAB2
74*1ABBA2
75*1ABCD2
76*A12ABC **
77*A12BAC **
78*A12BCA **
79*A12BBC **
80*A12BCB **
81*A12BCC **
82*A12AAB **
83*A12ABA **
84*A12BAA **
85*A12BBB **
86*A12AAA **
87*A12ABB **
88*A12BAB **
89*A12BBA **
90*A12BCD **
91*A1A2BC **
92*A1B2AC **
93*A1B2CA **
94*A1B2BC *
95*A1B2CB **
96*A1B2CC *
97*A1A2AB **
98*A1A2BA **
99*A1B2AA **
100*A1B2BB *
101*A1A2AA **
102*A1A2BB ***
103*A1B2AB **
104*A1B2BA *
105*A1B2CD **
106*A1AB2C *
107*A1BA2C *
108*A1BC2A *

109*A1BB2C *
110*A1BC2B *
111*A1BC2C *
112*A1AA2B *
113*A1AB2A *
114*A1BA2A *
115*A1BB2B *
116*A1AA2A *
117*A1AB2B *
118*A1BA2B *
119*A1BB2A *
120*A1BC2D *
121*A1ABC2 **
122*A1BAC2 **
123*A1BCA2 **
124*A1BBC2 **
125*A1BCB2 **
126*A1BCC2 **
127*A1AAB2 **
128*A1ABA2 **
129*A1BAA2 **
130*A1BBB2 **
131*A1AAA2 **
132*A1ABB2 **
133*A1BAB2 **
134*A1BBA2 **
135*A1BCD2 **
136*AA12BC *
137*AB12AC *
138*AB12CA *
139*AB12BC *
140*AB12CB *
141*AB12CC *
142*AA12AB *
143*AA12BA *
144*AB12AA *
145*AB12BB *
146*AA12AA *
147*AA12BB *
148*AB12AB *
149*AB12BA *
150*AB12CD *
151*AA1B2C *
152*AB1A2C *
153*AB1C2A *
154*AB1B2C *
155*AB1C2B *
156*AB1C2C *
157*AA1A2B *
158*AA1B2A *
159*AB1A2A *
160*AB1B2B *
161*AA1A2A *
162*AA1B2B *
163*AB1A2B *
164*AB1B2A *
165*AB1C2D *
166*AA1BC2

167*AB1AC2
168*AB1CA2
169*AB1BC2 **
170*AB1CB2 *
171*AB1CC2 *
172*AA1AB2
173*AA1BA2 **
174*AB1AA2 *
175*AB1BB2 *
176*AA1AA2
177*AA1BB2
178*AB1AB2
179*AB1BA2
180*AB1CD2
181*AAB12C *
182*ABA12C *
183*ABC12A *
184*ABB12C *
185*ABC12B *
186*ABC12C *
187*AAA12B *
188*AAB12A *
189*ABA12A *
190*ABB12B *
191*AAA12A *
192*AAB12B *
193*ABA12B *
194*ABB12A *
195*ABC12D *
196*AAB1C2 *
197*ABA1C2 *
198*ABC1A2 *
199*ABB1C2 *
200*ABC1B2 *
201*ABC1C2 *
202*AAA1B2 *
203*AAB1A2 *
204*ABA1A2 *
205*ABB1B2 *
206*AAA1A2 *
207*AAB1B2 *
208*ABA1B2 *
209*ABB1A2 *
210*ABC1D2 *
211*AABC12 *
212*ABAC12 *
213*ABCA12 *
214*ABBC12 *
215*ABCB12 *
216*ABCC12 *
217*AAAB12 *
218*AABA12 *
219*ABAA12 *
220*ABBB12 *
221*AAAA12 *
222*AABB12 *
223*ABAB12 *
224*ABBA12 *

225*ABCD12
226*112AAB
227*121AAB
228*122AAB
229*112ABA
230*121ABA
231*122ABA
232*112ABB
233*121ABB
234*122ABB
235*112AAA
236*121AAA
237*122AAA
238*112ABC
239*121ABC
240*122ABC
241*11A2AB
242*12A1AB
243*12A2AB
244*11A2BA
245*12A1BA
246*12A2BA
247*11A2BB
248*12A1BB
249*12A2BB
250*11A2AA
251*12A1AA
252*12A2AA
253*11A2BC
254*12A1BC
255*12A2BC
256*11AA2B **
257*12AA1B **
258*12AA2B **
259*11AB2A **
260*12AB1A **
261*12AB2A **
262*11AB2B **
263*12AB1B **
264*12AB2B **
265*11AA2A **
266*12AA1A **
267*12AA2A **
268*11AB2C **
269*12AB1C **
270*12AB2C **
271*11AAB2 **
272*12AAB1 **
273*12AAB2 **
274*11ABA2 **
275*12ABA1 **
276*12ABA2 **
277*11ABB2 **
278*12ABB1 **
279*12ABB2 **
280*11AAA2 **
281*12AAA1 **
282*12AAA2 **

283*11ABC2 **
284*12ABC1 **
285*12ABC2 **
286*1A12AB
287*1A21AB
288*1A22AB
289*1A12BA
290*1A21BA
291*1A22BA
292*1A12BB
293*1A21BB
294*1A22BB
295*1A12AA
296*1A21AA
297*1A22AA
298*1A12BC
299*1A21BC
300*1A22BC
301*1A1A2B
302*1A2A1B
303*1A2A2B *
304*1A1B2A
305*1A2B1A
306*1A2B2A
307*1A1B2B *
308*1A2B1B *
309*1A2B2B *
310*1A1A2A
311*1A2A1A
312*1A2A2A *
313*1A1B2C
314*1A2B1C
315*1A2B2C *
316*1A1AB2
317*1A2AB1
318*1A2AB2
319*1A1BA2
320*1A2BA1
321*1A2BA2
322*1A1BB2
323*1A2BB1
324*1A2BB2
325*1A1AA2
326*1A2AA1
327*1A2AA2
328*1A1BC2
329*1A2BC1
330*1A2BC2
331*1AA12B *
332*1AA21B *
333*1AA22B *
334*1AB12A *
335*1AB21A *
336*1AB22A *
337*1AB12B *
338*1AB21B *
339*1AB22B *
340*1AA12A *

341*1AA21A *
342*1AA22A *
343*1AB12C *
344*1AB21C *
345*1AB22C *
346*1AA1B2 ***
347*1AA2B1 ***
348*1AA2B2 ***
349*1AB1A2 ***
350*1AB2A1 ***
351*1AB2A2 ***
352*1AB1B2 ***
353*1AB2B1 ***
354*1AB2B2 ***
355*1AA1A2 ***
356*1AA2A1 ***
357*1AA2A2 ***
358*1AB1C2 ***
359*1AB2C1 ***
360*1AB2C2 ***
361*1AAB12
362*1AAB21
363*1AAB22
364*1ABA12
365*1ABA21
366*1ABA22 *
367*1ABB12 *
368*1ABB21 *
369*1ABB22 *
370*1AAA12
371*1AAA21
372*1AAA22 *
373*1ABC12
374*1ABC21
375*1ABC22 *
376*A112AB **
377*A121AB **
378*A122AB *
379*A112BA **
380*A121BA **
381*A122BA ***
382*A112BB *
383*A121BB *
384*A122BB *
385*A112AA **
386*A121AA **
387*A122AA *
388*A112BC **
389*A121BC **
390*A122BC *
391*A11A2B *
392*A12A1B *
393*A12A2B *
394*A11B2A *
395*A12B1A *
396*A12B2A *
397*A11B2B *
398*A12B1B *

399*A12B2B *
400*A11A2A *
401*A12A1A *
402*A12A2A *
403*A11B2C *
404*A12B1C *
405*A12B2C *
406*A11AB2 **
407*A12AB1 **
408*A12AB2 **
409*A11BA2 **
410*A12BA1 **
411*A12BA2 **
412*A11BB2 **
413*A12BB1 **
414*A12BB2 **
415*A11AA2 **
416*A12AA1 **
417*A12AA2 **
418*A11BC2 **
419*A12BC1 **
420*A12BC2 **
421*A1A12B *
422*A1A21B *
423*A1A22B *
424*A1B12A *
425*A1B21A *
426*A1B22A *
427*A1B12B *
428*A1B21B *
429*A1B22B *
430*A1A12A *
431*A1A21A *
432*A1A22A *
433*A1B12C *
434*A1B21C *
435*A1B22C *
436*A1A1B2
437*A1A2B1 **
438*A1A2B2 **
439*A1B1A2 **
440*A1B2A1 **
441*A1B2A2 *
442*A1B1B2 *
443*A1B2B1 *
444*A1B2B2 *
445*A1A1A2 **
446*A1A2A1 **
447*A1A2A2 *
448*A1B1C2 **
449*A1B2C1 **
450*A1B2C2 *
451*A1AB12 *
452*A1AB21 *
453*A1AB22 *
454*A1BA12 *
455*A1BA21 *
456*A1BA22 *

457*A1BB12 *
458*A1BB21 *
459*A1BB22 *
460*A1AA12 *
461*A1AA21 *
462*A1AA22 *
463*A1BC12 *
464*A1BC21 *
465*A1BC22 *
466*AA112B *
467*AA121B *
468*AA122B *
469*AB112A *
470*AB121A *
471*AB122A *
472*AB112B *
473*AB121B *
474*AB122B *
475*AA112A *
476*AA121A *
477*AA122A *
478*AB112C *
479*AB121C *
480*AB122C *
481*AA11B2 *
482*AA12B1 *
483*AA12B2 *
484*AB11A2 *
485*AB12A1 *
486*AB12A2 *
487*AB11B2 *
488*AB12B1 *
489*AB12B2 *
490*AA11A2 *
491*AA12A1 *
492*AA12A2 *
493*AB11C2 *
494*AB12C1 *
495*AB12C2 *
496*AA1B12 *
497*AA1B21 *
498*AA1B22 *
499*AB1A12 *
500*AB1A21 *
501*AB1A22 *
502*AB1B12 *
503*AB1B21 *
504*AB1B22 *
505*AA1A12 *
506*AA1A21 *
507*AA1A22 *
508*AB1C12 *
509*AB1C21 *
510*AB1C22 *
511*AAB112 *
512*AAB121 *
513*AAB122 *
514*ABA112 *

515*ABA121 *
516*ABA122 *
517*ABB112 *
518*ABB121 *
519*ABB122 *
520*AAA112 *
521*AAA121 *
522*AAA122 *
523*ABC112 *
524*ABC121 *
525*ABC122 *
526*1112AA
527*1121AA
528*1211AA
529*1222AA
530*1122AA
531*1212AA
532*1221AA
533*1112AB
534*1121AB
535*1211AB
536*1222AB
537*1122AB
538*1212AB
539*1221AB
540*111A2A
541*112A1A
542*121A1A
543*122A2A
544*112A2A
545*121A2A
546*122A1A
547*111A2B
548*112A1B
549*121A1B
550*122A2B
551*112A2B
552*121A2B
553*122A1B
554*111AA2
555*112AA1 **
556*121AA1
557*122AA2 **
558*112AA2
559*121AA2
560*122AA1
561*111AB2
562*112AB1 **
563*121AB1
564*122AB2 **
565*112AB2
566*121AB2
567*122AB1
568*11A12A
569*11A21A **
570*12A11A *
571*12A22A *
572*11A22A

573*12A12A
574*12A21A
575*11A12B
576*11A21B **
577*12A11B *
578*12A22B *
579*11A22B
580*12A12B
581*12A21B
582*11A1A2 **
583*11A2A1 **
584*12A1A1 **
585*12A2A2 **
586*11A2A2 **
587*12A1A2 **
588*12A2A1 **
589*11A1B2 **
590*11A2B1 **
591*12A1B1 **
592*12A2B2 **
593*11A2B2 **
594*12A1B2 **
595*12A2B1 **
596*11AA12 **
597*11AA21 **
598*12AA11 **
599*12AA22 **
600*11AA22 **
601*12AA12 **
602*12AA21 **
603*11AB12 **
604*11AB21 **
605*12AB11 **
606*12AB22 **
607*11AB22 **
608*12AB12 **
609*12AB21 **
610*1A112A *
611*1A121A *
612*1A211A *
613*1A222A *
614*1A122A *
615*1A212A *
616*1A221A *
617*1A112B *
618*1A121B *
619*1A211B *
620*1A222B *
621*1A122B *
622*1A212B *
623*1A221B *
624*1A11A2
625*1A12A1
626*1A21A1 ***
627*1A22A2 ***
628*1A12A2
629*1A21A2
630*1A22A1

631*1A11B2
632*1A12B1
633*1A21B1 ***
634*1A22B2 ***
635*1A12B2
636*1A21B2
637*1A22B1
638*1A1A12
639*1A1A21
640*1A2A11
641*1A2A22 *
642*1A1A22
643*1A2A12
644*1A2A21 *
645*1A1B12
646*1A1B21
647*1A2B11
648*1A2B22 *
649*1A1B22
650*1A2B12
651*1A2B21 *
652*1AA112 *
653*1AA121 *
654*1AA211 *
655*1AA222 *
656*1AA122 *
657*1AA212 *
658*1AA221 *
659*1AB112 *
660*1AB121 *
661*1AB211 *
662*1AB222 *
663*1AB122 *
664*1AB212 *
665*1AB221 *
666*A1112A *
667*A1121A *
668*A1211A *
669*A1222A *
670*A1122A *
671*A1212A *
672*A1221A *
673*A1112B *
674*A1121B *
675*A1211B *
676*A1222B *
677*A1122B *
678*A1212B *
679*A1221B *
680*A111A2 **
681*A112A1 **
682*A121A1 **
683*A122A2 *
684*A112A2 ***
685*A121A2 **
686*A122A1 *
687*A111B2 **
688*A112B1 **

689*A121B1 **
690*A122B2 *
691*A112B2 ***
692*A121B2 **
693*A122B1 *
694*A11A12 *
695*A11A21 *
696*A12A11 *
697*A12A22 *
698*A11A22 *
699*A12A12 *
700*A12A21 *
701*A11B12 *
702*A11B21 *
703*A12B11 *
704*A12B22 *
705*A11B22 *
706*A12B12 *
707*A12B21 *
708*A1A112 *
709*A1A121 *
710*A1A211 *
711*A1A222 *
712*A1A122 *
713*A1A212 *
714*A1A221 *
715*A1B112 *
716*A1B121 *
717*A1B211 *
718*A1B222 *
719*A1B122 *
720*A1B212 *
721*A1B221 *
722*AA1112 *
723*AA1121 *
724*AA1211 *
725*AA1222 *
726*AA1122 *
727*AA1212 *
728*AA1221 *
729*AB1112 *
730*AB1121 *
731*AB1211 *
732*AB1222 *
733*AB1122 *
734*AB1212 *
735*AB1221 *
736*11112A
737*11121A
738*11211A
739*12111A *
740*12222A *
741*11122A
742*11212A
743*11221A
744*12112A
745*12121A
746*12211A *

747*12221A
748*12212A *
749*12122A *
750*11222A *
751*1111A2
752*1112A1
753*1121A1
754*1211A1
755*1222A2
756*1112A2
757*1121A2
758*1122A1
759*1211A2
760*1212A1
761*1221A1
762*1222A1
763*1221A2
764*1212A2
765*1122A2
766*111A12 **
767*111A21 **
768*112A11 **
769*121A11 **
770*122A22 **
771*111A22 **
772*112A12 **
773*112A21 **
774*121A12 **
775*121A21 **
776*122A11 **
777*122A21 **
778*122A12 **
779*121A22 **
780*112A22 **
781*11A112 **
782*11A121 **
783*11A211 **
784*12A111 *
785*12A222 *
786*11A122 **
787*11A212 **
788*11A221 **
789*12A112 *
790*12A121 ***
791*12A211 **
792*12A221 *
793*12A212 **
794*12A122 *
795*11A222 *
796*1A1112 *
797*1A1121 *
798*1A1211 *
799*1A2111 *
800*1A2222 *
801*1A1122 *
802*1A1212 *
803*1A1221 *
804*1A2112 *

805*1A2121 *
806*1A2211 *
807*1A2221 *
808*1A2212 *
809*1A2122 *
810*1A1222 *
811*A11112 *
812*A11121 *
813*A11211 *
814*A12111 *
815*A12222 *
816*A11122 *
817*A11212 *
818*A11221 *
819*A12112 *
820*A12121 *
821*A12211 *
822*A12221 *
823*A12212 *
824*A12122 *
825*A11222 *
826*111112
827*111121
828*111211 **
829*112111 **
830*121111 *
831*122222 *
832*111122
833*111212
834*111221
835*112112
836*112121 **
837*112211 **
838*121112 *
839*121121
840*121211
841*122111 *
842*122221 *
843*122212 **
844*122122 *
845*121222 *
846*112222 *
847*111222
848*112122 **
849*112212
850*112221 *
851*121122
852*121212
853*121221 *
854*122112 *
855*122121 *
856*122211
```

Figure 9—Elimination of redundant test patterns due to device symmetry

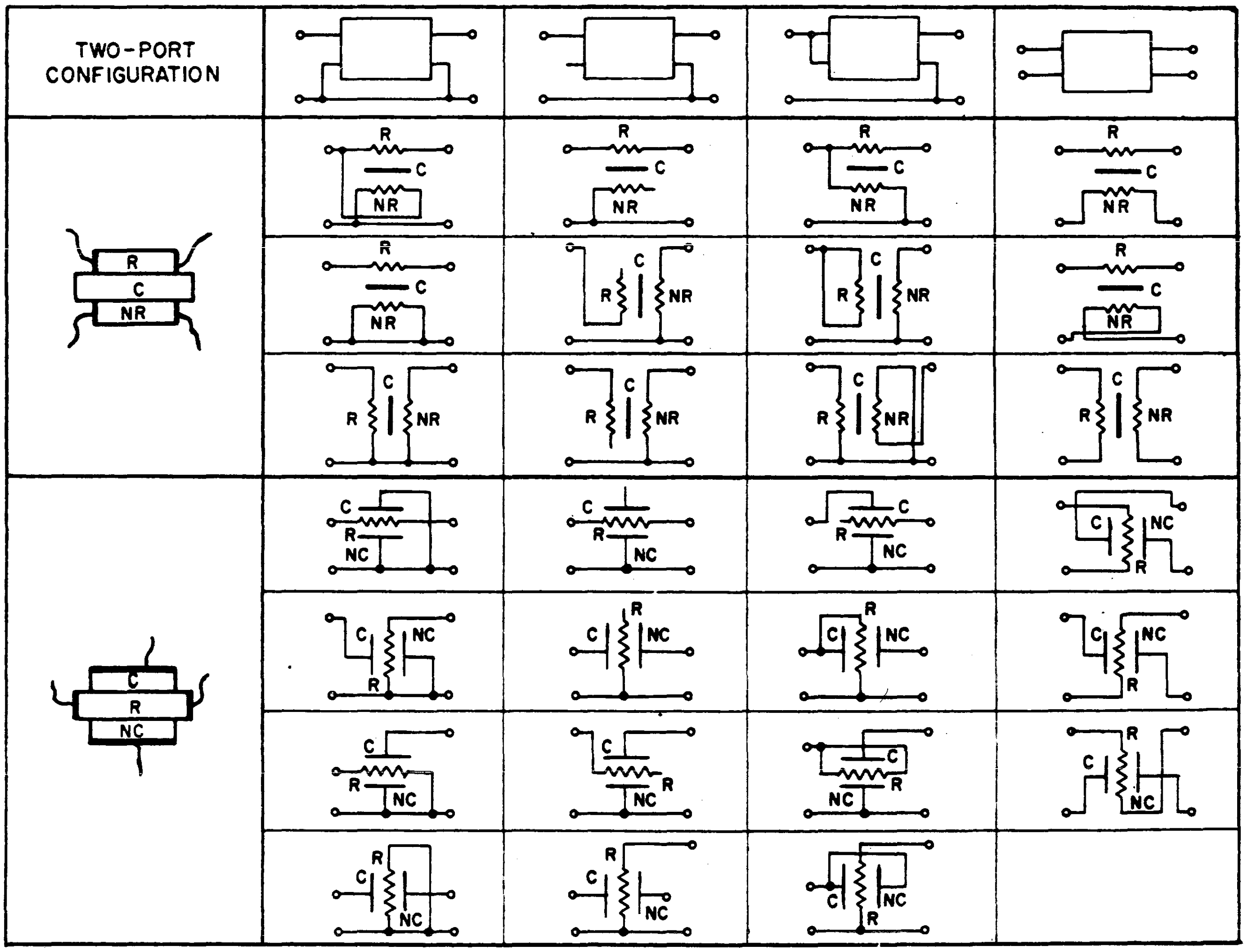


Figure 10—Unique test patterns of symmetrical device

December 1967 93-99 Also published in tutorial form, E. Sarkisian and W. W. Happ
Combinatorial techniques for fault identification in multiterminal devices
Proceedings Annual Symposium on Reliability January 1968 477–485

3 W W HAPP F J McINTOSH
Program SEAL-Subnetwork Enumeration and Listing
Available from Project COSMIC University of Georgia Computer Center, Athens Georgia as summarized in NASA Tech Brief #68-10227. Also F. J. McIntosh, Jr.
A computer program for subnetwork enumeration and listing
Proceedings Third NASA Microelectronics Conference Feb 1968 649–670

4 Proceedings IBM Scientific Computing Symposium on Combinatorial Problems Library of Congress Card Number 66–19006 March 1964

5 E F BECKENBACH (editor) D H LEHMER
The machine tools of combinatorics
Applied Combinatorial Mathematics John Wiley and Sons New York 1966

6 W W HAPP
Flowgraph techniques for closed systems
IEEE Transactions AES 2 No 3 May 1966 252–264

7 W W HAPP
Flowgraphs as a teaching aid
IEEE Trans on Education Vol E–9 No 2 June 1966 99-79

8 W W HAPP
NASAP: Present capabilities of a maintained program
IEEE International Convention Record Part 5 Circuit Theory March 1967 64–88

9 W W HAPP
Identification of test points in devices with specified symmetry
Proceedings International Symposium on Computer-Aided Design Southampton England April 1969

10 E F THOMAS
Effective screening of integrated circuits
Proceedings Third NASA Microelectronics Conference February 1968 327–334

11 E F THOMAS
DC pin-to-pin testing of integrated circuits
NASA/GSFC Paper 515–001 May 1967

12 L MAH L BUCHSBAUM T J B HANNON

Investigation of fault diagnosis by computation methods for microcircuits
Final Report Contract AF 33(615)-2094 November 1965

13 E RIVERA E R GARCIA R RANALLI
Computer generated fault isolation procedures
Proceedings 1967 Symposium on Reliability January 1967

14 Reliability Abstracts: Reviews pertaining to the following abstracts:
R66-12514
R66-12856
R67-13263
R68-13902

OS-3: The Oregon State open shop operating system

by JAMES W. MEEKER, N. RONALD CRANDALL, FRED A. DAYTON, and G. ROSE

Oregon State University Computer Center
Corvallis, Oregon

INTRODUCTION

This paper is a discussion of the OS-3 operating system developed at Oregon State University. Before proceeding to a discussion of that system, it is appropriate to say a few words in order to view this work within a more global context.

It is little more than a truism to say that computers are difficult and expensive to use. That is to say, computers are difficult and expensive with respect to the problems men wish to solve. One primary reason for this state of affairs is embedded in many years of cultural history. In the absence of computing machinery we have developed methodologies that ingeniously avoid the necessity for computational solutions.

For example, if it is necessary to perform several million multiplications in order to test a hypothesis, then until recently it was quite likely that such a hypothesis would remain unexplored. Now, of course, this situation is radically changed. Nevertheless, the thinking that will take proper advantage of current computer power is still in its infancy. Thus, computers are difficult and expensive to use because we haven't yet learned how to use them.

If we could see clearly enough into the future to determine those approaches to problem solving that will be most successful during the coming decades, then we would not hesitate to develop software tailored to best underwrite these approaches. To the degree that we lack the foresight to proceed in this fashion, there remains an acceptable alternative: the general purpose computer utility. If we can provide a utility that is inexpensive, reliable, and convenient to use, then we can deliver a powerful tool directly into the hands of the problem solver.

One overall requirement implicit in this idea is that such a utility must be comprehensive enough to free the problem solver from the burden of becoming a system's programmer. This burden has been responsible for the migration of many scientists from other disciplines into systems software development after which they proceed to neglect half a lifetime of training in their own field: an expensive proposition.

In addition, our objectives in developing a computer utility include the following: the utility should be

1. inexpensive—that is, system overhead should be small.
2. convenient—programming conventions should be easy to learn and use as well as generally accessible.
3. transparent—the user should have ready access to information about the state of the system, his account with the system, the status of a running program, and the contents of his saved storage.
4. information oriented—facilities must be available for creating, manipulating, and maintaining files of arbitrary structural complexity.
5. self extending—facilities should be available for building upon past experience in a facile way.

This paper is specifically a discussion of a time-sharing operating system that is intended to satisfy the first three of these objectives. The remaining two are also under development at O.S.U., but do not exist at the level of the resident operating system.

The following sections include a description of the system as seen by a user, a brief discussion of salient software characteristics, and a summary of system performance.

User features

OS-3 is a time-sharing operating system for the Control Data 3300 computer. It was developed at O.S.U.

and is presently our principal operating system. Currently the system can service up to 32 on-line users together with a batch user, and during early 1969, it is anticipated that this number will exceed fifty.

The system is used by various departments on campus, local industry, and other colleges throughout the State of Oregon. At the present moment the system logs approximately 4,000 console hours per month.

The operating system multiplexes available hardware resources among concurrent users (CRT, Teletype, and batch) in a time-slicing fashion. Processor time and core memory are allocated to running programs based upon considerations of program demand and page traffic flow.

Hardware environment

In order to orient those not immediately familiar with the 3300, it should be sufficient to say that the machine includes the following features:

- 24-bit word
- Paged memory and page file
- Executive mode operation
- Usual interrupt system
- Usual register configuration
- Real time clock
- 64-word fast core

Memory is expandable in units of 2^{14} words; we possess four such units. The 3300 addressing scheme will permit a user's program to address at most 2^{16} words. Our hardware configuration is depicted in Figure 1.

System library

The system library is composed of an absolute library and a user generated library. Most of the programs in the absolute library are written in reentrant code and treated as such by the operating system. The library includes:

1. Fortran compiler—a modified CDC Fortran with several input/output options including short form diagnostics suitable for listing at a console.
2. Algol compiler—modified CDC Algol.
3. OSCAR—an O.S.U.-developed conversational arithmetic interpreter with stored program capabilities.[1] OSCAR recognizes scalars, vectors, matrices; it is fully recursive and allows definition of functions and abbreviations. Formatting is allowed but due to the use of default format options it is not required. OSCAR can communicate with other available languages.

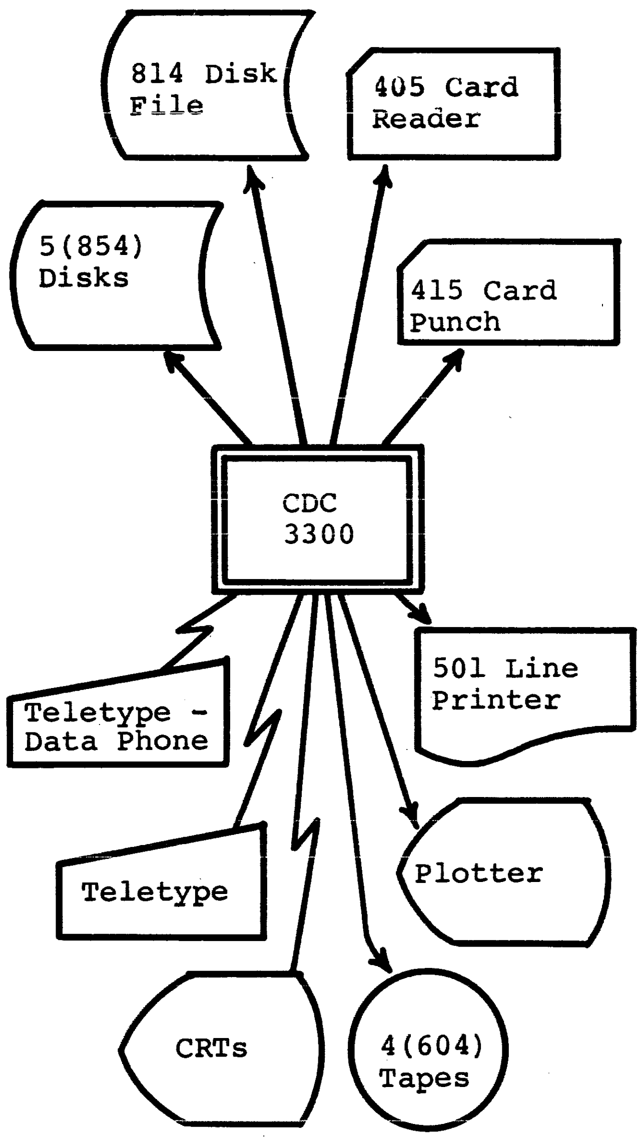


Figure 1—Hardware configuration

4. Compass—an extended version of the 3300 assembly language.
5. RADAR[2]—an on-line debugging language. RADAR includes an assembler/disassembler and permits single stepping through a program. (A CRT oriented version of the language is also available.)
6. Edit[3]—an on-line editing language with context searching capabilities.
7. Sort/Merge
8. Utilities—including Autoloading and file manipulation.

The user-generated library contains programs local to each user. If a user declares a program to be public, then anyone can access it in a fashion analogous to that used for the absolute library. Of course, only the creator can modify the program, and that only when it is not currently in use.

Input/output

At the level of the operating system, files have little structure. Two types of files exist: linear and random access. A *linear file* is physically a series of fixed length blocks. These are dynamically allocated up to some storage limit that is preset for each user. A user reads and writes a linear file in variable length records; writing a record in the middle of a linear file causes the remainder of that file to be released (see Figure 2).

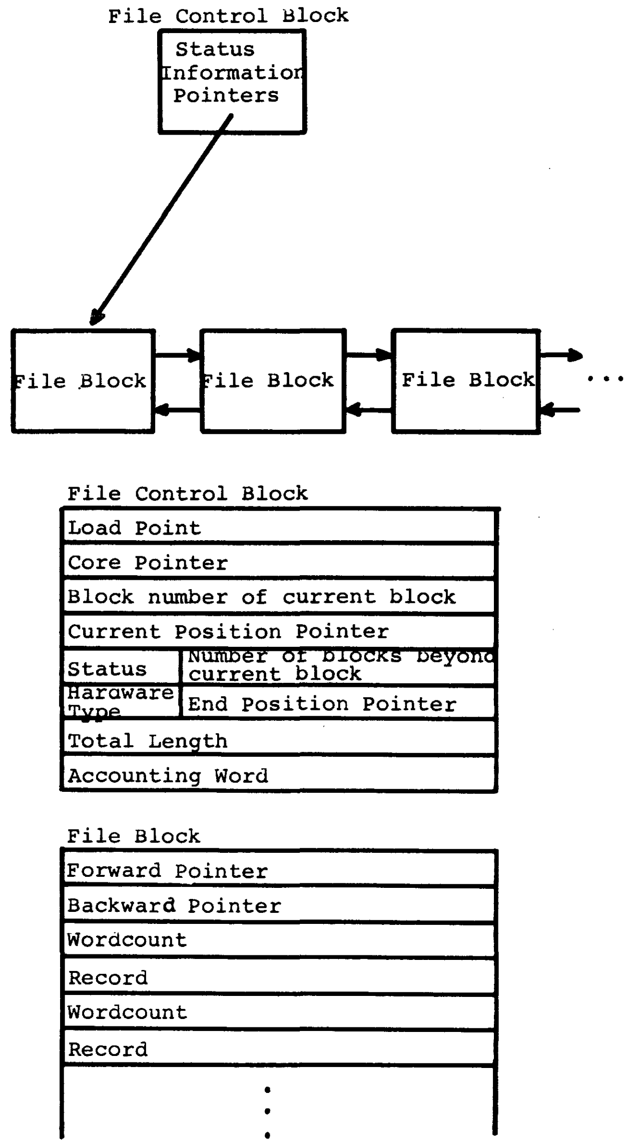


Figure 2—Linear file

Random access files can be viewed as terminally open files together with a file pointer. This type of file is essentially a large block of apparently contiguous storage (maximum size = $(2^9 - 2)2^{18}$ words). Physically the random access file is a series of fixed length blocks with a two level directory. The topmost directory holds pointers to a set of directory blocks, each of which contains pointers to a set of linked storage blocks. A serial pass through the file can use the links for traversing blocks, while a random search utilizes the directories (Figure 3). A user reads or writes a random access file n words at a time beginning at the current pointer location. At the end of the operation, the pointer is advanced by n words.

Files may be named and saved in semi-permanent storage. It is also possible to create temporary files that can be assigned to a logical unit number (in the range 0-99). Any saved file can be equated to a logical unit and conversely, a logical unit may be subsequently saved under a file name. While running programs manipulate logical units, not named files, the major programs in the library allow the user to supply only a file name that is then assigned to a logical unit by the program. Any file may be file protected. Any file may be placed in the public domain by preceding its name by an asterisk (*).

In addition to being equated to files, logical units may assume the following hardware types:

- line printer
- card reader
- card punch
- console input
- console output
- plotter
- magnetic tape
- null

Request processor

A console user is placed in control mode at the time he logs in, and he can revert to this mode at any time. Communication with the system while in control mode takes place via the *request processor*. In this language the user can call for any of the supported systems, execute jobs, manipulate files, examine the state of the system, the status of his running program, inspect information about his account, etc.

Of special note is that a user with a program in execution can cause that program to be suspended for an indefinite time by reverting to control mode. From control mode it is then possible to execute a sequence of commands and subsequently resume running by typing the command 'GO'. This capability may be used

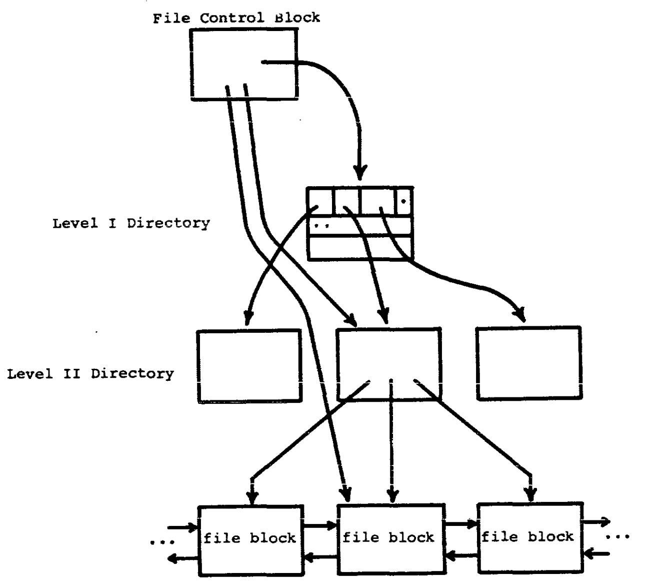


Figure 3—Random access file

to allow the user to solicit information about the running program or accomplish on-line error recovery. In a similar way, the command 'MI' simulates a manual interrupt that may be used for communication between the request processor and other supported systems.

Accounting

Associated with every user is a unique job/user number pair. The first number is used for billing purposes; the second is a code used for identification. Associated with each job/user number are three limits:

1. Maximum time—a time limit equal to the total processor time that may be used. This number is appropriately decremented at the conclusion of a session at the console.
2. Saved file space—a maximum storage limit for saved files.
3. Scratch file space—a maximum storage limit for temporary files.

When a user logs into the system, his time is automatically set to one minute and his scratch file limit to 100 storage blocks. He may change these limits up to the maximum limits associated with his job/user number.

When a console user logs off or when a batch job is completed, the charges for that job are converted to a CPU time equivalent and deducted from the user's remaining total time.

Charges are made for CPU time, some I/O, elapsed time at the console, and saved storage space. Since system overhead increases with increasing demand on the system, the apparent CPU time required for a job will be higher during peak hours. A user can get an indication of the current system loading by typing the command 'TRAFFIC'. If he decides against running and logs off at this point, no charge is incurred.

System characteristics

Several guidelines were adopted in writing the system. Development has been modular with all modules written strictly in tightly coded assembly language. Since core is at a premium in our configuration, it was imperative to keep the size of the resident monitor at a minimum. To this end, even the request processor is subject to memory swapping. In addition, the system is hardware sensitive, i.e., particular I/O instructions in the order code have been avoided because their failure rate, over the years, has been disproportionately high. Further, the system is highly parameterized. For example, if a memory module fails, a parameter can be changed and the system run in reduced memory.

With these guidelines in mind, the system can be described roughly as follows. Associated with every active user is a fixed length *program status area* (PSA) together with a set of attached lists. PSA's are linked together into a *user queue* (Figure 4). In general, OS-3 runs under the influence of this user queue, with resource allocation and I/O dependent upon the list structures connected to a particular PSA. At any given

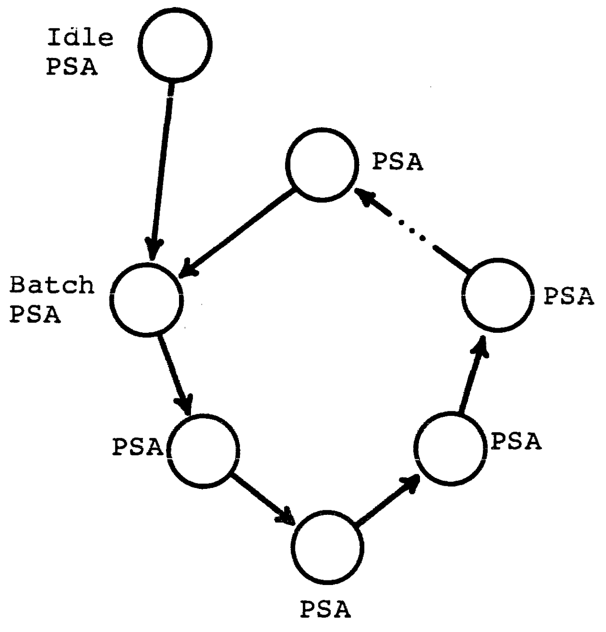


Figure 4—User queue

moment, the system is running a single user whose PSA is indicated by a pointer. Alteration of the contents of this pointer occurs at the end of discrete time intervals called quanta (see Figure 5). This general picture is qualified by other considerations as will be seen.

Physical memory is partitioned into four distinct areas:

1. The resident monitor
2. Free storage
3. File core blocks
4. Available memory

The resident monitor occupies approximately 10K words of core. PSA's with associated lists and console I/O occupy space in free storage that can be dynamically expanded. Two pages (2^{11} words per page) of file core blocks are reserved for disk transfers. The remaining core is assigned to running users and swapped. Any user can work in an address space of up to 2^{16} words of virtual memory.

The system includes the following major modules:

• Scheduler	—allocates processor time and memory
• User I/O	—interprets and handles user generated requests for I/O
• Intsort	—a recursive interrupt processor
• I/O Drivers	—a set of integrated device drivers
• Request Processor	—the command language interpreter
• Accounting	—user accounting routines

This paper will consider only the scheduler and User I/O in greater detail.

User I/O

User I/O is a collection of routines that supervise the peripheral device drivers and user requests for input or output operations. This module is divided into two principal sections: the mass storage device scheduler and the executive request interpreter.

The mass storage (M.S.) device scheduler governs a multi-priority transfer queue containing mass storage I/O requests. This queue is created by the M.S. device scheduler in response to requests received from peripheral devices in operation as well as user programs. For example, when the line printer driver exhausts its current block of output, and the next block is requested, the M.S. device scheduler interprets this request, assigns it a priority, and extends the transfer queue accordingly.

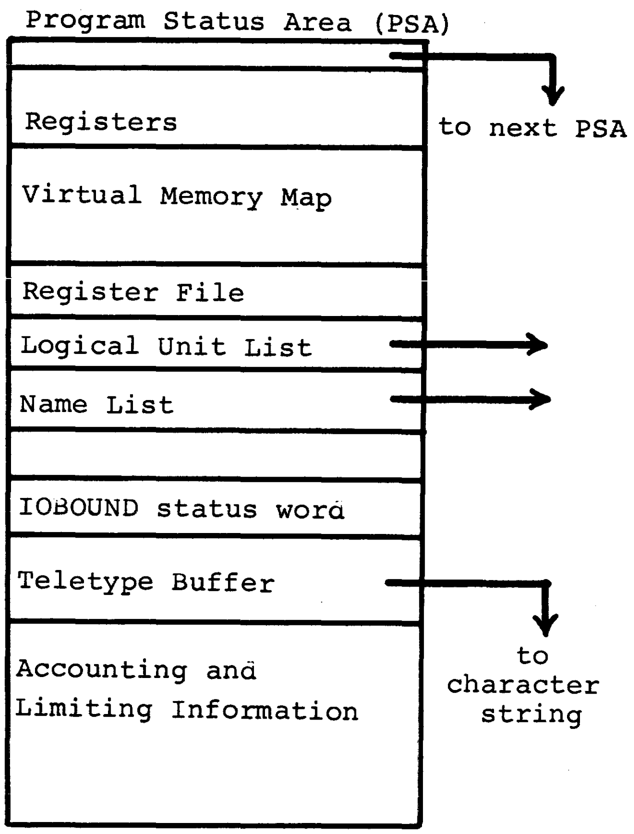


Figure 5—Program status area

The executive request processor interprets all executive requests (trapped instructions) generated by a user. After decoding a request, the processor delivers control to the appropriate routine. All I/O called for by a user is included in the category of executive requests.

Time and memory allocation

Allocation of the time and physical memory occurs primarily within the *Scheduler* and is controlled by the tables PAGETABLE and PAGETIME. Memory is divided into pages by the hardware. Each page has a page number associated with it which is used for relocation and reference to the software tables. Each page of physical memory has one word in PAGETABLE and a corresponding word in PAGETIME. Part of each PAGETABLE word serves as an indicator of the status of the associated page; the remainder is an address of a *page access word* which is used to map one page of virtual memory into the associated physical memory page. PAGETIME has one word per page that indicates the time at which a user last referenced the page in question. (See Figure 6.)

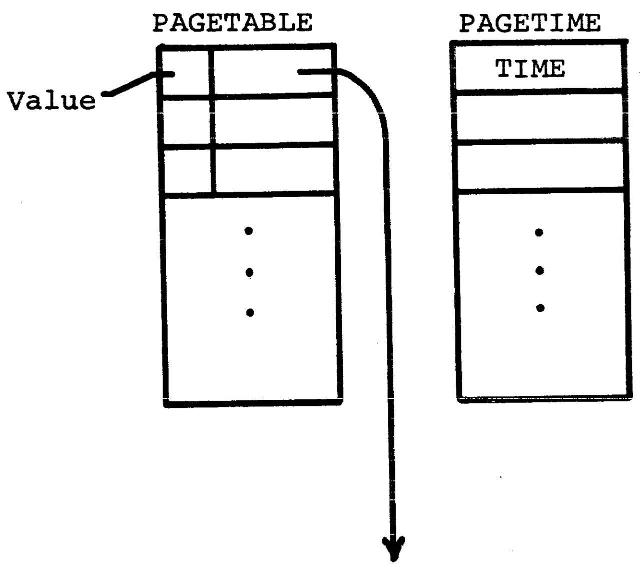


Figure 6—PAGETABLE and PAGETIME

When a page of memory is needed for swapping purposes, a search is performed on PAGETABLE to find the page with the smallest value. Value is infinite if the system bit is set or zero if the occupied bit is not set. In all other cases, a function is evaluated which is the sum of

$$\text{PAGETIME} - \text{clock} + \text{Value (I)} \qquad (1)$$

where

I	is a set of status bits from PAGETABLE
clock	is the current reading of the real time clock

and

Value	is given by a function that maps bit configurations into time values.

PAGETIME is set to clock + 1 hour whenever a page is referenced and is shifted right one position every hour to prevent overflow. The values in PAGETIME do not age linearly because of this shift; however, the function is continuous and is not inaccurate in the region where the clock is set back one hour and PAGETIME entries shifted.

A user's reference to his virtual memory occurs upon detection of an illegal write interrupt, and the system is then table driven based upon the above tables and the virtual memory map (VMM) in the user's PSA. Each word in the VMM is either a page access word or a pointer to a page access word, depending upon whether the page in question is written in reentrant code (Figure 7).

Processor time is also allocated to each user by the Scheduler. A running program pointer (RPSAPTR) advances in a circle around the user queue. If a user's program is to be activated when RPSAPTR advances to his PSA, then the following condition must be satisfied:

$$\text{TIMELEFT} > 0 \;\&\; \sim \text{IOBOUND} \qquad (2)$$

where

TIMELEFT—is the time left in the current quantum

and

IOBOUND —is a status word in the PSA that indicates whether the user is awaiting the completion of an I/O operation.

If no PSA satisfies condition 2, the RPSAPTR advances until a PSA that is not IOBOUND is found and sets that program to run another quantum.

It is clear that a user who requires extensive swapping will contribute significantly to the flow of page traffic

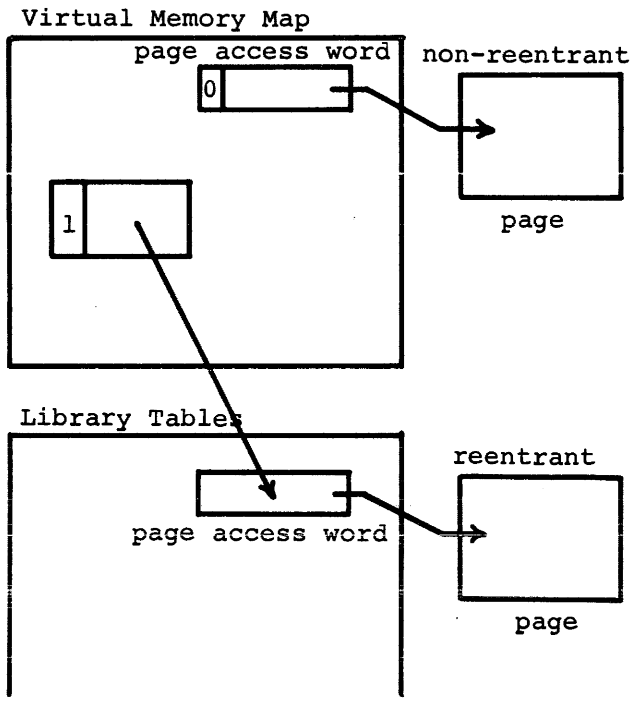


Figure 7—Page access word

to and from the disk. Moreover, such a user will receive poor service if the page traffic flow is heavy.

In order to cope with this situation, a more sophisticated scheduling arrangement is required. This kind of extension to the scheduling algorithm should be adaptive in nature, that is the system should recognize the existence of a problem situation and proceed to 'tune itself up.' Further, detection of the problem condition as well as modification of the scheduling tactics must be easily computable if an undesirable increase in system overhead is to be avoided.

At present, OS-3 includes an initial version of a demand scheduling strategy called the debogging algorithm. This algorithm governs the allocation of processor memory so as to minimize page traffic flow in the presence of varying user requests.

Conceptually, the algorithm can be viewed as a high priority pointer that is cycled around the user queue independent of the RPSAPTR. If a user is designated the high priority user, then his in-core pages increase in value, and he is automatically placed second in line in the swapping queue for requested pages.

In effect, the debogging strategy tends to delay users whose page requests are heavy with respect to current page traffic flow, and then run such users with greater priority for a period of time during which they can occupy a substantial amount of core.

In particular, the algorithm governs the behavior of the following independent categories of events:

1. Advancing the high priority pointer
2. Delaying troublemakers, and
3. Rehabilitating former troublemakers.

These categories are now described in greater detail.

The high priority pointer, HPP, is advanced to the next user whenever one of the following conditions is satisfied:

$$\text{The user logs off} \tag{3}$$

$$\text{The user becomes I/O bound} \tag{4}$$

$$WCT + (PR_i * 2^{K1}) \geq K2 \tag{5}$$

where

WCT —is the amount of wall clock time that has elapsed since the HPP was advanced

PR_i —is the page request word in the ith user's PSA. PR_i is incremented by one each time the ith user requests a page

and

K1, K2—are constants.

K1 influences the rate at which the HPP will shift to the next user if the current user is busy swapping. K2 is simply a constant that governs the cycle rate of the HPP.

If any of the conditions 3, 4, or 5 is satisfied, then

$$WCT: = \overline{SQ}: = QCT: = 0 \tag{6}$$

$$PR_i: = 0 \quad (i = 1, \ldots, n) \tag{7}$$

where

$\overline{SQ}$ —is the average length of the swapping queue

and

QCT —is the number of useful quantums of computing (i.e., time not spent in the idle loop).

The second category concerns the troublemaker. A user is a *troublemaker* if only the mass storage wait bit of his IOBOUND word is set, and if

$$PR_i \geq K3 \tag{8}$$

where

K3 —is a constant that determines the number of swap requests a user must generate in order to qualify as a troublemaker.

The *last troublemaker* is defined as the troublemaker that is located the greatest distance from the HPP in the direction of pointer rotation. A troublemaker is delayed by setting the delay bit in the IOBOUND word of his PSA. The last troublemaker will, in fact, be delayed if:

$$(SQ \geq K4) \ \& \ (\overline{SQ} \geq K5) \tag{9}$$

where

SQ —is a counter that contains the current length of the swapping queue

$\overline{SQ}$ —the average length of the swapping queue

K4 —is a stabilizing factor

and

K5 —determines heavy page traffic flow.

If condition (9) is satisfied then

$$SQ: = \overline{SQ}: = QCT: = 0 \tag{10}$$

The final category provides for the rehabilitation of former troublemakers. A *former troublemaker* is a user whose delay bit is set. The *closest former troublemaker* is defined to be the former troublemaker located the least distance from the HPP in the direction of pointer rotation. The closest former troublemaker may be rehabilitated by clearing his delay bit. This will occur when

$$(\mathrm{QCT} \geq \mathrm{K6}) \ \& \ (\overline{\mathrm{SQ}} < \mathrm{K7}) \qquad (11)$$

where

K6 —is a stabilizing factor

and

K7 —determines relatively light page traffic flow.

The effect of the preceding debogging strategy is to match available processor memory to user demands. If this cannot be done, then an obvious troublemaker is delayed, and, after a period of stabilization, the situation is sampled again to determine whether an acceptable match has occurred. If not, then another troublemaker is delayed, and so forth, until a match is achieved. Conversely, if user demands are not overloading the swapping queue, then former troublemakers are rehabilitated, one at a time. Of course, if several users require large quantities of physical memory, the recidivism rate will be high.

System performance

System performance measured in terms of system overhead tends to be quite good. If the total number of user hours for a month is compared to the total amount of billable CPU time for that period, it turns out that the system spends slightly more than 65 percent time in an idle loop. Of course, this might indicate that the system is heavily I/O bound; however, test measurements indicate that this is not the case.

In another test, switching time was measured by loading the system with a sample job mix. Jobs were chosen from three categories:

1. Compute bound
2. 65K swap bound
3. I/O bound

The 100 millisecond quantum was then reduced until no useful computing took place. This break-even point occurred at four milliseconds.

ACKNOWLEDGMENTS

The authors are indebted to Steven K. Sullivan for his incisive comments, useful criticisms, and system programming support.

REFERENCES

1 J DAVIS
A brief description of OSCAR (Second Revision)
OSU Computer Center cc-68-45

2 J MEEKER
RADAR
OSU Computer Center cc-68-30

3 F DAYTON W MASSIE
OS-3 teletypewriter editor manual (Revised)
OSU Computer Center cc-68-17

Virtual memory management in a paging environment

by NORMAN WEIZER and G. OPPENHEIMER

Radio Corporation of America
Camden, New Jersey

INTRODUCTION

The Spectra 70/46 Time Sharing Operating System (TSOS) is designed to be a combined time-sharing and multiprogramming system that will support up to 48 conversational users or a combined total of 64 batch and interactive tasks processing simultaneously.

The memory management subsystems of TSOS maintain complete control of main core memory, the drum backing store and the virtual memory facilities of the entire system. The virtual memory management subsystem controls the allocation and release of the backing store space, the organization of the 2 million byte virtual memory and the characteristics (the control-bit settings) of the allocated virtual memory space.

Hardware description

A short description of the relevant spectra 70/46 Processor[1] features is presented here to provide a background for the discussion of the virtual memory management subsystem. The 70/46 is basically identical to the spectra 70/45 Mod II Processor[2] with the addition of a flip-flop implemented hardware translation memory. The dynamic translation facilities of the 70/46 are provided by this translation memory and the special functions implemented in the read only memory.

The translation memory (TM) contains 512 half word elements each of which represents a single virtual page. The page size used within TSOS is 4096 bytes, and thus the virtual memory is a linear space[3] of two million bytes.

Each half word element in the translation memory is composed of a set of control bits and a physical page number, shown in detail in the Appendix. The control bits indicate whether the page has been modified, whether it has been accessed, whether it may be modified, whether access is restricted to privileged users and whether the page is in memory. If a referenced page is in memory the physical page number is used in conjunction with the 12-bit displacement field of the virtual address to determine the physical address. If the page is not in memory the hardware generates a paging queue interrupt, and the software, utilizing a hardware special analysis function, determines the page (s) required and causes the page (s) to be brought into main memory.

The 24 bit virtual address format is shown in Figure 1. It represents the address formulated after all address arithmetic has been performed.

The Page and Displacement portions of the virtual address constitute the 18-bit address field and are generated by the 18-bit address arithmetic. If the sum of the least significant 18 bits of the base register, an index register, and a displacement field of an instruction would normally cause a carry into the 19th bit of the address field, this carry is lost and an address wrap around to the lower boundary of a segment takes place, thus providing a modified form of segmentation.

The segment, unused, and D bit fields of an address can be changed in the base registers by using the normal binary addition capabilities of the processor. The D bit is used to obtain direct (untranslated) addressing capability while in the 70/46 or translation addressing mode.* Only privileged Control system functions can use this facility.

The nine bit field formed by the segment and page bits of the virtual address forms the index which is used to determine which of the 512 translation memory elements corresponds to the addressed virtual page.

* The Spectra 70/46 is also capable of being run in a 70/45 mode. In the 70/45 mode no address translation takes place and the address space is limited to 262K bytes.

1 bit	2 bits	3 bits	6 bits	12 bits
D	Unused	Segment	PAGE	Displacement

Figure 1—Virtual address format

The six page bits contained within the indexed TM entry (see Appendix) are concatenated with the 12 low order bits of the virtual address to form the 18 bit physical address actually used by the processor to address memory.

Two memory protection capabilities are provided in the 70/46. The first capability is provided by a set of protection key locations associated with main core memory. These keys are only used in the 70/45 mode of processing, although they are also operational in the 70/46 mode. The second capability is provided by the translation memory implementation and is only available when in the 70/46 mode. A nonprivileged routine in the 70/46 addressing mode, or a privileged function not using the direct (untranslated) addressing capability, cannot address information unless a translation memory element for that task allows translation to that memory location. In this way, unless the entries for two users are simultaneously loaded into translation memory, no user can access the private information of another user. Also, the control bits of the translation memory entries prevent nonprivileged access to unauthorized information and also prevent modification of code which is executable only.

The backing store for the 70/46 is a fixed head drum of either 800 or 1600 tracks. The track capacity of the drum is approximately 5000 bytes. By assigning a single 4096 byte page per track, the 3600 RPM drum can accommodate 60 page transfers per second. The time to transfer a page between core and drum is approximately 13.65 msecs, thus leaving about 3 msecs-free time between the end of one page transfer and the beginning of the next. This free time (gap time) is an upper bound on the amount of processing that may be performed between page transfers if the full drum transfer rate capability is to be realized.

All of the I/O operations, including the paging transfers, use untranslated or direct addresses. This requires that the virtual to physical address conversion must be made before an I/O is initiated. Also, any pages involved in an I/O operation, including those which contain the I/O control information, must remain in core during the duration of the I/O operation.

Paging algorithm

A demand type paging algorithm[3] is implemented in TSOS.[4] This algorithm limits the number of tasks simultaneously competing for the processor and main memory by using a "working set"[5] like concept in the scheduling of tasks.

When a task in made "active" (i.e., is allowed to compete for processor time and main memory) the counter of available main memory pages is decremented to set aside the number of pages it is anticipated the task will require. This number is equal to the number of pages used by the task during its previous activation period.

Rather than fully swapping a program's working set into memory or allocating specific memory pages for the task at activation time, however, only those pages required by the task's first instruction are actually pre-paged into core. During a task's active period, its pages in core are normally considered nonpageable.* When a task is deactivated, the counter of available main memory pages is incremented, and all of the task's pages in core are placed on the page-out queues.

When a task is blocked by a paging queue interrupt, pages are chosen from the page-out queues and the appropriate drum transfers are initiated. During the period in which the required pages are being brought into core, other active tasks are placed in control of the processor.

Memory management design considerations

In general a memory management subsystem for a multi-access system should have the following characteristics:

1. Protection—no user should be able to destroy the data belonging to another user or to the system as a whole;
2. Privacy—without authorization, no user should be able to access the data belonging to another user or the private system data;
3. Shared Code Use—several users should be able to simultaneously use the same physical copy of commonly used routines or programs;
4. Flexibility—the full memory management capabilities provided by the hardware, consistent with the protection and privacy considerations, should be made available to the user programs;
5. Ease of Usage—the memory management facilities should be provided to the user in a manner which allows them to be easily used;
6. Low Overhead—the use of the memory man-

* An exception to this rule occurs if a single task requires more pageable main memory space than is available in the system.

agement facilities should add as little overhead to the system as possible, consistent with the other characteristics;

7. Integrity of Design—the memory management subsystem should not be designed as a unit separated from the remainder of the operating system. It must be designed as an integral part of the overall system but with clearly defined boundaries and interfaces. The clearly defined boundaries and interfaces prevent a great many problems in the implementation and debugging phases of operating system development. (The method used to develop the scheduling and paging algorithms for TSOS is described in Reference 4.)
8. Modularity—the memory management subsystem should be designed as a set of modular routines. There should be simple and sharply defined interfaces between the various routines to simplify implementations and debugging problems.

In the following sections a description of the TSOS Virtual Memory Management Subsystem is provided and an attempt is made to show how all of the above criteria were met within the hardware environment described above.

Virtual memory organization

The two million bytes of virtual memory are divided into two equal units. Each user of the system is permitted to use the first one million bytes for code and data areas related strictly to his own task. The second one million bytes are reserved for Control System functions and shared code.

In terms of the use of the translation memory this means that the first 256 entries are used for private user task information. Each time a new task gains control of the processor the previous task's translation memory entries are stored in main memory, and the new task's entries are loaded into the translation memory. During this entire process the upper 256 entries in the translation memory are unchanged.

This organization of the virtual memory, aside from reducing the overhead entailed by the loading and unloading of the translation memory, permits the Control System to be written using virtual addressing and at the same time to have full access to all user areas. Since the task in control of the processor has its entries loaded into the translation memory while it is running, the task's memory is directly available to the Control System through the translation mechanism. (The converse is not true, in that the Control Program pages are privileged and the pages containing shared code are executable only, preventing user code from accessing Control Program information and from modifying shared code.)

If the virtual memory were not divided as it has been, and the full two million bytes had been made available to each user, the Control System would have had to use direct addressing to a much greater degree and would have required much more code to be resident, or the loading and unloading of the translation memory would have been appreciably greater.

Backing store allocation

Although each user task has a private one million byte virtual memory, the memory is not actually usable until it is dynamically allocated by means of the memory management macros; that is, until a realtionship is set up between a page of virtual memory and a page of backing store. In a conventional processor this is analogous to saying that the address space (which is normally equal to the physical memory size), is not usable until the program is loaded into memory. And then only the assigned portion of the total address space (memory) may be referenced.

Within TSOS user pages are allocated when a program is loaded and when additional space is dynamically requested. When a page is allocated, a translation memory entry is initialized for it and a drum track is assigned. This track is associated with the page on a permanent basis, i.e., until the page is released and the translation memory entry is no longer valid.

The relationship between the backing store track and the page of virtual memory is maintained even while the page is in main memory for the following reasons. The number of pages of main memory is small compared to the number of pages on the backing store. Therefore, the marginal gain in drum tracks available to the system through the use of a dynamic assignment system would be small. General utilization of the drum tracks in this manner would also increase the probability of binding the system intolerably should the drum become saturated.

From another viewpoint, the fact that there is only a single page per track means that schemes which reassign drum tracks to core pages that must be written out, so as to optimize drum utilization in a multiple page per track environment, are not applicable in the environment of TSOS.

In summary, until a virtual page is requested and backing store assigned to it, the virtual memory space it represents is not usable. Any attempt to access an unallocated page is detected by a combination of hardware and software and is treated as a program addressing error.

Virtual memory classification

To regulate the use of virtual memory, and to simplify its request, particularly within the Control System, virtual memory is divided into six somewhat arbitrary classes. The address assignments for the six classes are shown in Figure 2. The characteristics of each class are described below.

Class 1 Virtual Memory is occupied by the resident portion (kernel) of the Control System. All Class 1 pages are privileged and nonpageable. There are no drum images of these pages. At present there are 10 Class 1 pages in TSOS.

Class 2 Virtual Memory is occupied by the nonresident portion of the Control System. All Class 2 pages are privileged and pageable and may be marked as executable only, depending upon the nature of the routines occupying them. There is a drum image for each of these pages.

Classes 1 and 2 virtual memory are preallocated at system generation. The boundary between these two classes (C1LIM) may be varied from system to system dependent upon installation requirements.

Class 3 Virtual Memory is occupied by the dynamically acquired resident portion of the Control System. All Class 3 pages are privileged and nonpageable. There are no drum images of these pages. This memory class is used for task control blocks, terminal I/O buffers and certain system work space. It is also dynamically released when the requirement for resident space lessens.

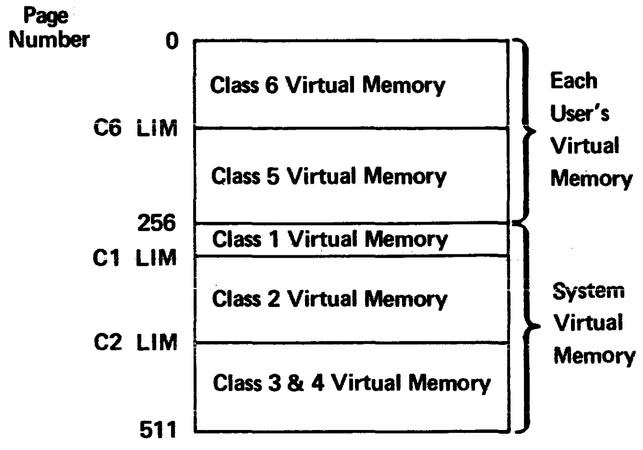


Figure 2—Virtual address space assignments of virtual memory classes

Class 4 Virtual Memory is occupied by the nonresident work space dynamically acquired by the Control Program and by the shared code called by the users of the system. All Class 4 pages are pageable and have drum images. The Class 4 pages used by the Control System are marked privileged, but those used for shared code are marked nonprivileged.

Virtual memory Classes 1 through 4 constitute the system virtual memory. As a group these four classes must be contained within the one-million bytes of address space available to the system. They reside in the upper one-half of the translation memory and are not changed (swapped) in the translation memory as control is passed from user to user.

Virtual memory Classes 5 and 6 constitute the user's virtual memory. Together these two classes are limited to the one million bytes available to the user. They occupy the lower one-half of the translation memory and as control is passed from user to user the Class 5 and Class 6 translation memory entries for each user are swapped out of and into the translation memory. This means that any data stored in a user's Classes 5 and 6 Virtual Memory cannot be accessed using virtual addresses when that user's entries are not loaded into TM. (This, in turn, means that the system must use direct, non-translated, addressing to access user memory for a user that is not in control of the processor.)

Class 5 Virtual Memory is occupied by dynamically allocated pageable areas acquired for the specific user by the Control System. These pages may be marked privileged or nonprivileged. They are used for task dependent information such as task dependent virtual memory tables, protected file control blocks, program loader data, data maintained by the interactive debugging language and I/O buffers acquired for the task by the system.

Class 6 Virtual Memory is occupied by dynamically allocated pages acquired by the user for his code and work areas. The pages of Class 6 memory are under control of the user task.

The boundary between Class 5 and 6 memory (C6LIM) is completely variable and depends upon the requirements of each individual task. Normally Class 5 memory occupies the 16 pages from page 240 through 255, and Class 6 memory occupies the 240 pages from page 0 through 239. Each memory class is allocated contiguously such that a page of Class 5 memory is never bounded on both sides by pages of Class 6 memory or vice-versa.

Shared code

Nonprivileged shared code offers the potential advantages of savings of main memory and backing store space plus a reduction in the paging rate. However, additional memory management control logic is required to realize these advantages. In systems with true segmentation, the segment is normally the unit which is shared and shared code may be used by attaching the called segment to the virtual memory of the calling task. This degree of generality in a system with a linear address space requires more control logic than the potential advantages warrant.

With a linear address space it seems preferable to allocate some of the address space for shared code and to take this space out of the system's area of virtual memory. This procedure eliminates the need for any overhead producing special actions when a task using shared code gains or loses control of the processor. It also permits the same algorithm to be used for paging the Control System and the shared code, simplifying the design and implementation of the paging subsystem and thus reducing system overhead.

The major disadvantage of this approach is that the (virtual memory) space for the shared code must be allocated for every user, whether or not he uses the shared code. However, it is felt that the low overhead, ease and flexibility of use, and ease of implementation more than make up for the loss of some address space.

In TSOS the system administrator determines for his specific installation what major routines will be considered eligible for sharing and makes this determination known to the system by means of a special command. He may choose only RCA supplied software such as the File Editor, and the Interactive Fortran compiler; or some user designed programs; or any combination of the two. Upon the first call for one of these shared routines the loader allocates the amount of memory needed to load this routine. This memory is allocated as nonprivileged, execute only, Class 4 virtual memory. Upon succeeding calls for the same routine, the loader establishes links between the shared routine and the calling task without the need for reloading the shared program in any form.

During execution each user of the shared routine uses the same physical (and virtual) copy of the routine as all other users.

Macro calls

The acquisition and release of virtual memory and the control of the characteristics of allocated virtual memory are the major services performed for users and other Control System functions by the virtual memory management subsystem. These services are requested by means of macros which generate standardized linkages and parameter lists. These linkages may be either Supervisor Call instructions (SVCs) or standardized branching conventions, both of which provide clean interfaces, an invaluable aid in the debugging phases of complex system development.

The macros are named REQM (request memory), RELM (release memory), and CSTAT (change memory status). There are two forms of each macro, one which may be used by nonprivileged and privileged (Control System) routines, and the second which is restricted to privileged routines only.

The nonprivileged forms of the REQM and RELM macros permit the user to request and release Class 6 memory in multiples of one page, with a maximum of 64 pages per call. If the address spaces and backing store space is available, the requested memory will be allocated in the first unallocated area (lowest available area in the address space) large enough to satisfy the request; or if the user so specifies, the memory will be allocated starting at a specific address.

The nonprivileged form of the CSTAT macro allows the user to change the status of any page in Class 6 memory to read-only or read-write. The CSTAT macro also provides the mechanism for users to request that specific Class 6 pages be made pageable or nonpageable.*

The privileged forms of the virtual memory macros allow Control System routines to operate on any page in Classes 2, 3, 4, 5 and 6 virtual memory. The option of the CSTAT macro which changes a page's status to read-only or read-write is available for all memory classes. The option to make pages pageable or nonpageable is available only for memory Classes 2, 4, 5 and 6. This option of the CSTAT macro is the most heavily used as it permits the Control System to lock into (unlock from) main memory pages which are (were) required to be resident for I/O operations.**

The privileged forms of the REQM and RELM macros permit Control System functions to request and release Classes 3, 4 and 5 virtual memory. Classes

* Provision exists within the system, in certain well defined situations to permit users to use this option of the CSTAT macro. The limit of the number of pages that a user may make nonpageable is established based upon system-wide parameters and conditions set at task initiation.

** All I/O is done with nontranslated addressing and thus commands must contain physical addresses and buffers must not be moved until the I/O operations complete.

1 and 2 virtual memory are structured at system generation. In addition to the full page allocation capability of the nonprivileged version of the macros, partial page allocation is provided in the privileged versions.

Partial page allocation

Many Control System functions require different sized areas of memory during their execution. This memory may be required specifically for a single task or it may contain system wide information. Memory space which need not always be resident and which is required for a single task is acquired as Class 5 memory; system wide information which is pageable is stored in Class 4 memory and user dependent or system dependent information which is nonpageable is stored in Class 3 memory.

To conserve address space, better utilize main memory and reduce the paging rate for Control System pages, Classes 3, 4 and 5 memory are allocatable in partial page units. The units of allocation are 8n bytes where $2 \leq n \leq 509$.

Any request for larger size areas are allocated in full page increments. Any size area may be requested, but during the allocation process the size allocated is rounded up to the next larger standard size. This standardization, making all allocations multiples of a single quantum size, eases both the allocation and garbage collection processes employed.

Each page allocated is treated as a separate unit so that no partial page allocation crosses a page boundary. This serves two purposes. First it eases the record keeping involved by limiting the number of areas considered in a single operation. Second it prevents dynamically acquired I/O buffers from being allocated across page boundaries.

The latter is significant in that otherwise it would be necessary to page contiguous virtual pages into contiguous main memory pages, and this would vastly complicate the paging and physical memory management subsystems.

To manage the partial page allocation two linked lists are maintained in each subdivided page. One list, termed the main list, links all of the areas on the page in address order. The second list, termed the free list, links all of the unallocated areas in area size order, with the smallest area at the head of the list. The links of both lists are eight bytes long. The entries in the links include a free bit, which is used to indicate unallocated areas, a size field, forward and backward link fields and a two byte integrity field used by software to check that the link was not destroyed by some other software routine.

In addition to the memory links, partial page tables are also maintained by the system to manage partial page allocation. Two of the tables are maintained in Class 3 memory to control the Class 3 and Class 4 memory partial page allocations. There is also a corresponding partial page table in each user's Class 5 memory which is used to control the Class 5 partial page allocations for that user. The entries in each table are identical. They consist of the virtual page number of the page to which they correspond and the size of the largest free area on the page. There is one entry for each page which is subdivided for partial page allocation.

The placement of the memory links on the same page as the partial page areas presents the possibility of malfunctioning system components destroying the links. However, rather than proving to be a hindrance, this link placement proved to be a great aid in system debugging. This is due to the fact that the memory management routines will often be the first system function to find the destroyed link. This, in turn, helps to avoid the problem that some other system function will malfunction, because it uses an adjacent area which was also destroyed, allowing many bugs of the type which would only occur at widely scattered intervals to be more easily tracked down.

Memory management tables

A relatively complex table structure is required to support the memory management functions of TSOS. These tables support the physical memory management and paging subsystems along with the virtual memory management subsystem. They are used primarily to maintain allocation status information for the major memory resources—the core pages, the drum pages, the system virtual address space and user virtual address space.

The allocation status information for drum pages and for system virtual memory pages is maintained in bit-per-page maps called the *Paging Drum Memory Map* and the *System Virtual Memory Map*. These tables are used when the request memory (REQM) macro code must find an unallocated drum page during the allocation of a page of pageable virtual memory and when it is necessary to determine the address of free pages during the allocation of system virtual memory. These tables are also used during the corresponding RELM (release memory) processing.

The core status data are maintained in two tables called the *Physical Memory Map* and the *Physical Page Allocation Table*. Each entry of the Physical Memory Map indicates whether the page is free or allocated, the memory class data for nonpageable pages and certain reservation information. The Physical Page Allocation Table contains the drum address (for pageable pages), the I/O count (the number of I/O operations in process or scheduled into this virtual page), link space for the page out queues, and the address of the *Virtual Page Table* entry for pageable pages.

The *System Virtual Page Table* is a two part table. The main portion contains the core image of the entries loaded into the translation memory for the system virtual memory. However, when the pages represented by these entries are not in core, the cylinder portion of the backing store address is maintained in these entries. The secondary part of the table is used to store the drum track portion of the backing store address.

The above described tables are maintained in Class 1 virtual memory. They are system wide tables. In addition, there are four private tables maintained for each user. They are the *Block Address Table* and the associated *User Virtual Page Table* which are maintained in Class 3 Virtual Memory, and the *User Virtual Memory Map* and the *Class 5 Partial Page Table.*

The Block Address Table entries for each user are maintained within the Task Control Block (TCB). The TCB contains the master information about each task in the system. The Block Address Table entries of a task are used within a special function to cause the User Virtual Page Table entries to be loaded into the translation memory when the task is to be given control of the processor, and conversely when these entires are to be stored in core when control of the processor is removed from the task. The space used to store these entries is maintained in System Virtual Memory to guarantee their accessibility by the Control System at any time. Otherwise, they would be accessible only when the user was in control of the processor.

The User Virtual Memory Map parallels the System Virtual Memory Map and is allocated in the user's Class 5 memory. The Class 5 partial page table is also allocated in the user's Class 5 memory. It is used to control the partial page allocation of the user's Class 5 memory.

SUMMARY

The salient hardware features of the system described are: a linear address space of 512 pages of 4096 bytes each; a main memory of 64 pages; a single level page per track backing store of 800 or 1600 pages; and the use of a 512 entry translation memory to effect the virtual memory of the system.

The facilities controlled by the virtual memory management subsystem described include the organization of the virtual memory, the subdivision of the virtual memory into classes, the management of the shared code within the system, and the allocation of backing store and of partial pages.

Within the context of the hardware structure, the major aspects and advantages of the described software system are summarized below.

The partitioning of the virtual memory to concurrently accommodate the system and a single user reduces translation memory swapping overhead and provides the system code with full accessibility to all user code, while still permitting the system code to be written using virtual addresses.

The division of the virtual memory into classes structures the use of the virtual memory, regulates its use and simplifies the request and release procedures, especially within the Control System.

The incorporation of sharable code wit in the system virtual memory affords its direct accessibility to all users, permits a single page table to be maintained for the code, and allows the same paging algorithm to be applied to shared code as is used for system code; but it requires all users to give up the same amount of virtual memory for the shared code, whether or not they use the shared code.

The allocation of backing store only when a page of pageable virtual memory is allocated enables more users to be run concurrently with a given level of backing store than if the backing store was allocated for the entire user virtual memory, regardless of the user's ntent to utilize his entire virtual memory.

The maintenance of a relationship between a virtual memory page and a track on the backing store, even when the page is in memory, is justified based upon the real probability that the page may not be modified and therefore will not have to be written out—if the backing store association is maintained while it is in memory; and the added consideration that the ratio of drum tracks to memory tracks is such that the marginal gain in drum tracks available to the system from reassigning pages in memory is extremely small. The drum characteristic of a single page per track is also a factor in this regard.

The provision for partial page allocation for other Control System functions, while it increases the calls on the virtual memory subsystem, provides for better utilization of memory and easier development of reentrant code.

REFERENCES

1 ———
RCA Spectra 70/46 Processor Reference Manual
2 ———
RCA Spectra 70/35 45 55 Processor Reference Manual
3 B RANDELL C J KUEHNER
Dynamic storage allocation systems

C A C M Vol 11 No 5 May 1968 297–306
4 G OPPENHEIMER N WEIZER
Resource management for a medium scale time sharing operating system
C A C M Vol 11 No 5 May 1968 313–322
5 P J DENNING
The working set model for program behavior
C A C M Vol 11 No 5 May 1968 323–333

APPENDIX

The translation memory

The Translation Memory is 512 half-words in size. Each entry in Translation Memory has the format shown in Figure 3.

The meaning of each of the control bits and the physical page number in the translation memory entry is given below:

P = Parity bit (invisible to the software).

W = Written Into Bit: indicates when set, that the page addressed in memory by this translation halfword has been written into. This bit is automatically set by hardware and reset by software.

G = Accessed Bit: indicates, when set, that the page addressed in memory by this translation halfword has been accessed (read, or written into). This bit is automatically set and reset by hardware. Attempted but unsuccessful access to a page does not set this bit.

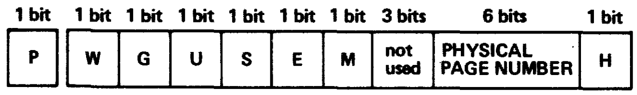


Figure 3—Format of a translation memory entry

U = Utilization Bit: indicates, when set, that the addressed translation word can be utilized. This bit indicates, when reset, that the addressed translation word cannot be utilized (i.e., this virtual page is not in core) and a Paging Queue Program Interrupt occurs. This bit is set and reset by software.

S = State Bit: Indicates when set, that the addressed translation word is nonprivileged. When this bit is reset, it indicates that the address page is privileged and can only be accessed by a program operating in the privileged mode (i.e., a portion of the system software). When this bit is reset and a nonprivileged program attempts to access this page, a Paging Error Program Interrupt occurs. This bit is set and reset by software.

E = Executable Bit: indicates when set, that the page addressed in memory by this translation word can be read as an operand or instruction but cannot be written into. When this bit is reset, all forms of access are allowed for this page. If a program attempts to write into a page with this bit set in the translation word, a Paging Error Interrupt occurs. This bit is set and reset by software.

M and H bits are used when the 2048 byte virtual page mode is used. Under TSOS only the 4096 byte virtual page mode is used.

Physical Page Number: when the U bit is set, these six bits contain the six most significant bits of the actual physical address of the page represented by this T.M. entry. The full physical address is obtained by concatenating these six bits with the low order 12 bits of the virtual address. When the U bit is reset no meaningful information is contained in this field.

An operational analysis of a remote console system

by HERBERT D. SCHWETMAN
and JAMES R. DELINE

The University of Texas
Austin, Texas

INTRODUCTION

The Computation Center of The University of Texas at Austin provides remote console access to a CDC 6600 computer through a system called RESPOND.[1] RESPOND was written by Control Data Corporation and has been in operation at The University for more than two years.

The paper gives a brief description of RESPOND and the capabilities provided the user. This is followed by a critical evaluation of the performance and reliability of the RESPOND system based upon experience gained in its use. A survey of user reactions is presented next. Finally, the cost of providing this remote batch entry service is estimated in terms of percent of system resources used and system maintenance required.

A description of RESPOND

The RESPOND system was installed on the CDC 6600 at the Computation Center on March 10, 1967, by the Special Systems Division, Control Data Corporation. This was the second implementation of RESPOND on a 6000 series computer,* and the first under the SCOPE 2.0 operating system. This version later became the framework for Control Data's standard 6000 series system—TTY RESPOND.[2]

A RESPOND terminal is typically a Model 33 or 35 Teletype, with or without punched paper tape capability. From this terminal a user may log into the system by providing his password and the account number to which his computing activities will be charged. He may then enter data into the system via the keyboard or paper tape and may create a file by giving this text a name. Such files then become a member of the user's private file catalog. Files may also be introduced into RESPOND from other sources. Card decks and magnetic tape records can be copied into files in the user's file catalog.

RESPOND appends a sequence number to each incoming line of text and, by referring to these numbers, the user is provided with a limited text-editing capability. A line or group of contiguous lines of text may be displayed at the terminal by referring to the name of the file and specifying the desired lines. New lines may be inserted into the body of the file at any point, and undesired lines may be deleted by reference to their sequence number. Two or more files may be merged, with the new file given a name different from the others.

At the user's option, these files may be copied to punched cards or printed output at the 6600 site, may be submitted as programs and data to be run by the 6600, or may be saved by the RESPOND system for use at a later time. Files which are no longer wanted may be deleted by the user from his file catalog; those files which remain are periodically dumped to magnetic tape. In addition to holding all user files, this tape holds public files, which are accessible as read-only files by all users, and a list of passwords for all authorized users. This tape is copied to disk storage each morning when RESPOND is placed "on the air" and is used to restore the system when RESPOND experiences an unrecoverable failure.

RESPOND files of program source text may be submitted for compilation and execution. In this environment, a RESPOND job consists of one or more RESPOND files, the first of which is a SCOPE control card file. The contents of this file are identical to the control card record which would be used if the program were to be run in the normal (over-the-counter) batch

* The first implementation was at Rechenzentrum der Technischen Hochschule Aachen, Aachen, Germany, on a CDC 6400 in January 1967.

environment of the 6600. Thus, the same compilers, assemblers, utility and object-time subroutines, deck structures, and error messages provided in the normal batch mode are available to the RESPOND user. As will be pointed out in a later section, there are some advantages and disadvantages to this feature.

Once a set of files have been submitted for execution, they are locked from further user access until the job has been run. The user is not able to interact with his program once it is placed into execution, but he is permitted to create, peruse, and submit for execution other files in his file catalog.

After the program has been run, RESPOND will collect those files specified by the user which were created as a result of a program execution. Typical of these files is the standard output file which ordinarily will contain listable output from a compiler and, in the case of a subsequent load-and-go, the results produced by the compiled program. These files are converted into RESPOND format and are placed in the user's file catalog. Other special-purpose files may be left with the operating system for on-site disposal, such as plot and microfilm files, or magnetic tapes created during program execution. A user/computer-operator message facility is available to permit close cooperation on magnetic tape requirements, log-out times, etc.

The commands of RESPOND can be divided into five groups. The frequency distribution of the usage of these commands is given in Table I. It is interesting to note that use of commands in the EDIT group far outweighs command usage in the other groups.

Table I—Command frequency distribution

EDIT group		61%	STATUS group		9%
Clear	10%		Flst (Fast List)	5%	
Delete	10		Status	3	
Load	9		List	1	
File	9		BATCH Group		9%
Show	9		Submit	6%	
Enter	8		Copy	2	
Display	6		Compile	1	
Format	0		Assemble	0	
UTILITY Group		12%	MISCELLANEOUS		9%
Login	6%		Break	7%	
Logout	3		Errors	2	
Save	2		Set	0	
Message	1				

3,775 Commands in Sample

The RESPOND system environment

The CDC 6600 computer at The University of Texas at Austin consists of a high-speed central processor with 131,072 60-bit words of central memory and 10 peripheral processors, each with 4,096 12-bit words. All memories have a 1.0 μsec cycle time; the central memory has 32 independent banks permitting an upper limit of 10 memory accesses per micro-second. The peripheral processors can read and write the central memory as well as their own private memories and may address any one of the twelve high-speed input-output channels. The central processor has no input-output instructions.

Jobs can be entered into the 6600 from card readers within the Computation Center, from any of five remote computers which communicate via broad-band telephone lines, and from RESPOND. On a typical weekday, about 2,500 jobs are processed by the central computer, of which some 300 originate at RESPOND consoles.

The SCOPE operating system at The University of Texas at Austin requires a resident of 13,000 words of central memory and occupies two of the ten peripheral processors. In addition, SCOPE will call upon the remaining eight peripheral processors from time to time to service users' I/O buffers in central memory, load jobs or library routines from the disk, service the card readers by placing incoming jobs into the input queue, schedule jobs for loading into central memory, and attempt to keep the print, punch, plot, and microfilm queues empty.

Central memory is dynamically broken into seven logical areas called control points. User programs are assigned to these control points for processing, with the central processor servicing the program in central memory of the highest priority which does not have any incomplete I/O buffers. In practice, one of these seven control points is required to service all of the unit record equipment; a second control point is occupied by the central processor portion of the RESPOND system. The central memory requirement for the former control point varies between 512 and 8,192 words, while the latter requires 14,600 words as a minimum and increases as a function of the activity at the remote terminals. The maximum available central memory for user programs is approximately 103,000 words.

Programs read from the card readers are placed in the input queue on the disk and are assigned a central memory access priority of two octal digits. The first digit varies inversely as the time limit requested and the second varies inversely as the central memory space requested. Both requests are extracted from the job card which is the first card of the control card record for the program. Jobs submitted from RESPOND terminals are constrained by policy to a time limit of 127 seconds and central memory limit of 32,768 words. Since 97 percent of all jobs run at the Computation Center run in less than 127 seconds, the

RESPOND time limit is not unduly restrictive. While these job card parameters could result in a modest input queue priority had the job been entered through a card reader, RESPOND assigns to all of its jobs a very high priority.

The central memory scheduling algorithm is based upon the following criterion: if a control point is available, the highest priority job in the input queue whose central memory request is less than or equal to the current unused central memory is brought to a control point. Once there, the job runs to completion, and its disposable files are collected and routed to the proper output devices. Since RESPOND jobs are given a very high input queue priority, they are normally assured of rapid assignment to a control point.

One peripheral processor is dedicated to servicing the RESPOND communication line multiplexer, which can accommodate up to 64 data sets. This peripheral processor polls all active lines eleven times per second, packing input characters into the appropriate central memory buffer and placing output characters on the line for transmission to the remote terminal. At the present time 15 AT&T 103A2 data sets are available through a rotary switching scheme. This dial-up feature permits optimum usage of the available modems and also permits a recorded audio message to be returned to the user if he should happen to dial up when RESPOND is inoperative.

In addition to the peripheral processor required to service the multiplexer, RESPOND occasionally requests other peripheral processors to assist in file merging, job submission, etc. Also, small percentages of certain transient system peripheral processors are required for job scheduling, job processing, and a dozen or so other system functions.

Evaluation of performance

The performance discussed in this section refers to physical characteristics of RESPOND/user interaction. The measurements made of this aspect of performance are of (1) system response, (2) user "think time," (3) delay in processing of jobs in the SCOPE input queue, and (4) a history of RESPOND reliability. The architecture of the 6600 makes possible easy access to these measurements, since the peripheral processors are independent of the central processor and may be called upon to monitor RESPOND's progress from time to time.[3]

The response time statistics were gathered by a subroutine in the multiplexer servicing program, which is resident in a peripheral processor. The processing delay statistics were gathered by post-processing of the SCOPE chronological log (called the DAYFILE). Every time a RESPOND job is submitted to the input queue, a DAYFILE message is generated. At some later time, the job is assigned to a control point and another DAYFILE message is issued. Since the time of issue is entered along with the message into the DAYFILE, elapsed time between these two events can easily be measured. The selection of the statistics was greatly influenced by similar studies of other remote console systems.[4,5,6]

The central processor portion of RESPOND is activated by SCOPE once every 500 milliseconds. During the few milliseconds it is active, RESPOND services all terminals which have completed an input message in the past one-half second. Generally, this servicing can be completed and a response placed in the terminal's output buffer in one "duty cycle"; however, several cycles may be required for the more complex commands.

Figure 1 is a graph displaying the probability density curve of the "system response time." This response time is defined to be the elapsed time between a user supplied carriage return (end of message) and the beginning of the first output character from RESPOND.

Figure 2 shows the probability density curve of "user response time" or "think time." Think time is defined to be the time between the system response or "go ahead indication" and the next user input character. Think time is not really a measure of system performance but is provided in order to allow system designers to see an example of user performance. The

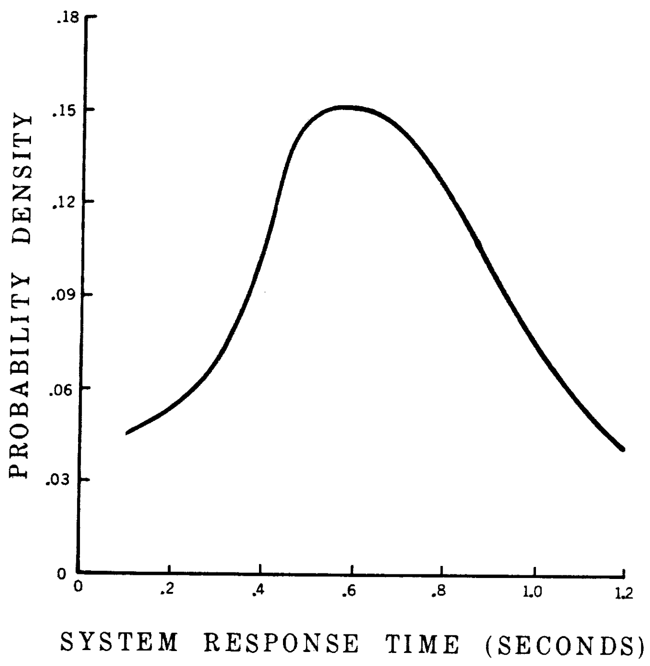


Figure 1—System response time

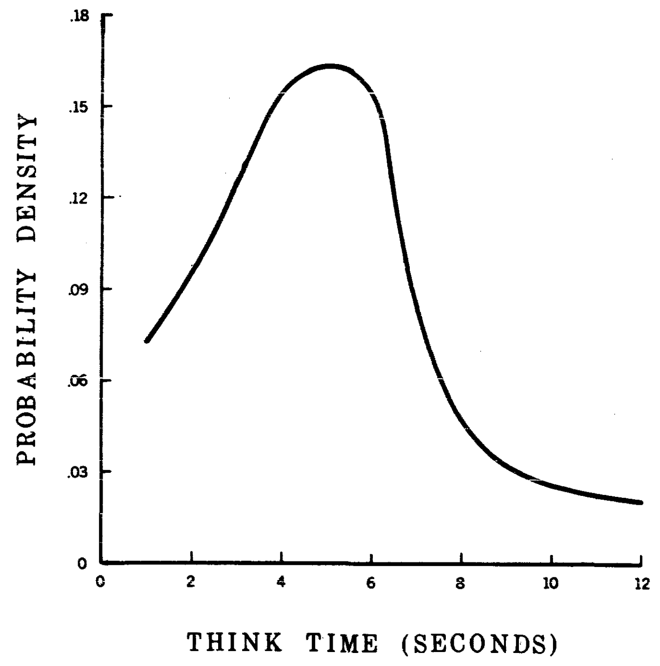


Figure 2—Think time

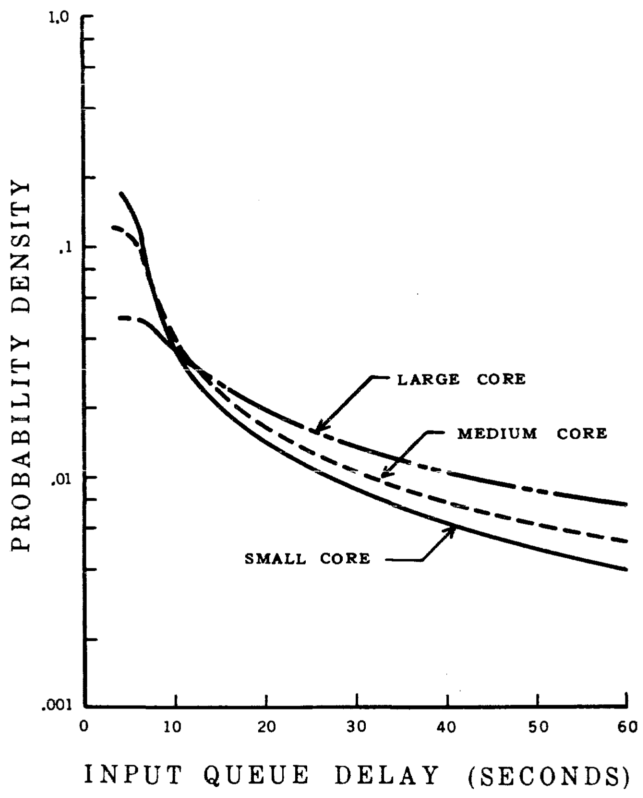


Figure 3—Input queue delay

"typical" RESPOND user is further characterized in Table II.

Table II—The typical RESPOND User[6]

Time at Console	15.25 minutes
Number of jobs submittted	1.12 jobs
Computer time	8.1 seconds
Number of commands input	19
Number of lines of data input	29
Number of lines of data output	71
Number of disk accesses	43
Average think time	5.5 seconds

Figure 3 indicates the input queue delays for submitted jobs. The small core curve represents the delay for jobs requiring between 0 and 8,192 words of central memory, the medium core curve for jobs between 8,193 and 24,576 words, and the large core curve for jobs between 24,577 and 32,768 words. These curves were derived from more than 28,000 RESPOND jobs observed over a 9-month period of time. Although these curves represent delays primarily due to a temporary unavailability of sufficient central memory in which to run the job, it can be seen that the operating system is rather insensitive to varying central memory requirements.

As was demonstrated in a survey of user reactions, the most important single attribute of a remote console system is system reliability. One of the most exasperating events in man-machine interaction is for the man to spend time keying in a text and then for the machine to "lose" it. Thus, in spite of almost instantaneous response time and a multitude of user-oriented conveniences, an unreliable system is of little value. RESPOND reliability has been uneven at best.

The types of system failures include RESPOND bugs, SCOPE failures which cause RESPOND to malfunction, and hardware failures. Currently, there is a restart capability which permits recovery from many of these failures. This restart permits a user's files created prior to his most recent SAVE command to be recovered. Thus all failures result in some loss of files, but in most cases, the losses are minimized. Other failures cause the loss of all files created since the time of the last dump of RESPOND files to magnetic tape which, in the worst case, is four hours.

Figure 4 is a history of all RESPOND failures from September 1967 to January 1969. The ordinate of the graph is the ratio of the number of failures per thousand RESPOND jobs submitted. It should be noted that this ordinate is a logarithmic scale. Currently, the ratio of unrecoverable failures (dump tape reload) to recoverable failures (restart) is about 1:10.

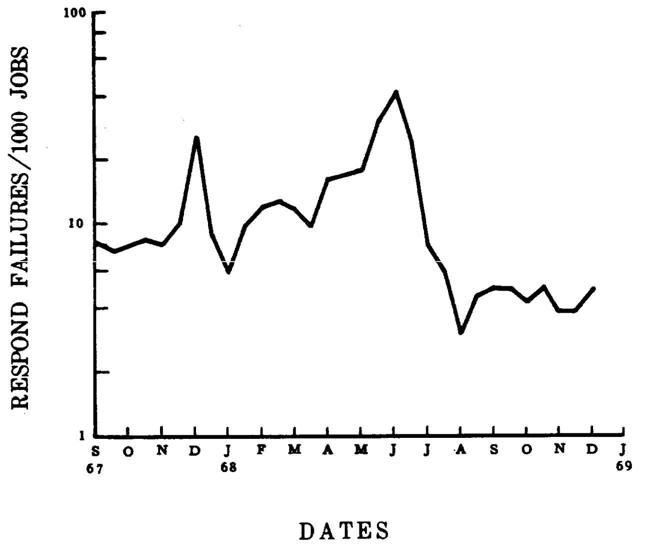


Figure 4—History of RESPOND failure rate

Certain system failures include those which involve a failure in the communication equipment. As initially installed, RESPOND made no check of the modem status. In some cases, a telephone line was inadvertently disconnected, but the user's password remained logically "in use." With the installation of the dial-up network, this created a serious problem, since a user could no longer select his point of connection to the multiplexer. A local modification added a test for "modem connected" status to the multiplexer servicing routine and automatically initiated a LOG-OUT if a disconnected modem was found. This modification has virtually eliminated all RESPOND failures due to communication equipment malfunctions.

User reactions

Initially, RESPOND passwords were issued only to faculty and staff personnel and selected graduate students. This was due in part to the novelty of the system and a feeling of a lack of need on the part of many potential users. More recently, the user population has grown to include graduate students in many disciplines and certain undergraduate classes as well. Presently, there are 142 active passwords outstanding with 1,194 files in their file catalogs.

A questionnaire was recently distributed to all RESPOND users. They were asked to comment on their usage of RESPOND, to give their opinions of the reliability of the system and to suggest improvements which they felt could be made to the system.

Complaints from the user population can be easily broken down into two groups: failures of RESPOND to do the things it should do, and improvements that could be made to the system. In the first category, the single biggest complaint was the unreliable nature of the system in saving user files from one session to the next. A new user quickly learns (usually the hard way) that newly created files should be copied to punch cards, magnetic tape, paper tape, or the printer as a precautionary measure. These files can then be re-introduced as RESPOND files with minor inconvenience to the user. Table III presents a tabulation of users' ratings of RESPOND's reliability.

Table III—Users' ratings of RESPOND's reliability

Rating	*Percent of Responses*
Excellent	2.3%
Good	18.2%
Fair	41.0
Poor	29.4
Unusable	9.1

Two highly desirable features of RESPOND are the control card compatibility with the batch system and the availability of the entire system library to the RESPOND user. These features eliminate duplication of programming and system maintenance in the applications software area and permit a user to easily switch between normal batch processing and RESPOND without fear of system incompatibility. A summary of programming languages utilized by users who answered the questionnaire is given in Table IV.

Table IV—Programming languages utilized by RESPOND users

Programming Language	*Percent of Users Responding*
FORTRAN	98%
ALGOL	22
LISP	22
COMPASS	20
Other	5
L6	2
SNOBOL	2

The improvements which were suggested included the implementation of a context-oriented text editor, provision for a conversational or interactive capability, and modification of some of the system-wide services to limit the amount of printed output for RESPOND users. The first and third suggestions are particularly important when the remote console in use has a slow-speed printer. The most consistently suggested improvement was that reliability and dependability be improved. The group making this suggestion stressed the idea that RESPOND is a useful tool in their work but that this usefulness would increase as system integrity and dependability improved.

Those who had the most praise for the system were those who otherwise would have had limited access to the 6600. They tended to have longer sessions at their terminals and were quite creative in their handling of multiple files. One graduate student claimed he completed his research for his doctoral dissertation a year early due to his access to a RESPOND terminal.

Cost

The cost of providing a remote console capability within a multiprogramming system is difficult to determine due to the following considerations:

- Often, the remote console system is just taking up "slack" in the system resources.
- The remote console system can ease the load on the normal batch processing portion of the computation center.

It cannot be denied that system interference, the occupation of valuable memory space and a control point, and the use of one dedicated peripheral processor by the remote console system represent tangible costs. Table V shows the utilization of various system resources over a period of one month.

Table V—RESPOND utilization of system resources

	Maximum	*Average*	*Minimum*
Characters of disk storage required for RESPOND files	—	14.5x10^6	—
Central memory words required by RESPOND	29,800	22,100	14,600
Percent of available central processor time used by RESPOND*	6.9	4.7	0.39
Percent of available peripheral processor time used by RESPOND*	18.1	14.1	12.5

The next two figures illustrate the demand made of certain system resources as a function of the number of

* These figures do not include time required to run the programs submitted from RESPOND terminals. The peripheral processor percentages include the dedicated peripheral processor which services the multiplexer.

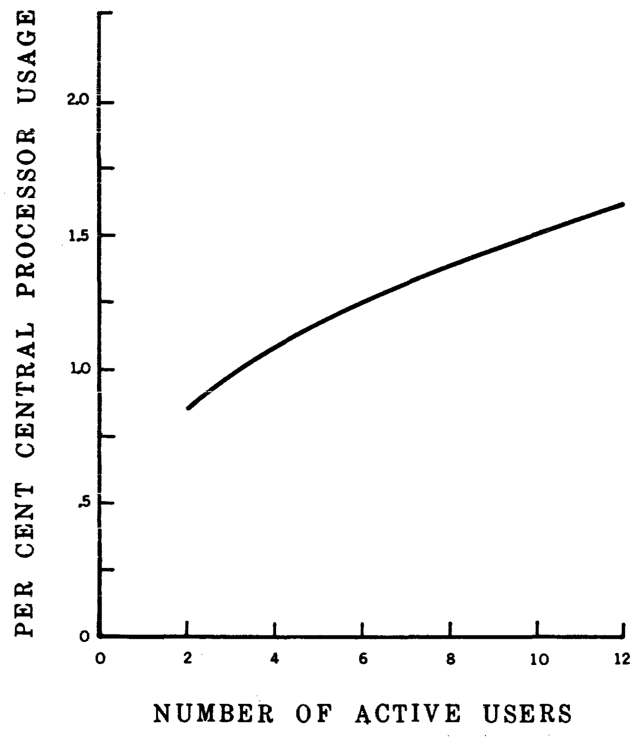


Figure 5—Central processor usage

active users. Figure 5 indicates central processor time used by RESPOND. This measurement does not include the periods of dump tape loading and unloading, which averaged 0.62 seconds per file. With no users logged in, the central processor portion of RESPOND requires 1.94 milliseconds to complete a duty cycle. Since duty cycles are initiated once every 500 milliseconds, an overhead of 0.39 percent of available central processor time is obtained. Figure 6 reflects RESPOND's requirements for additional central memory workspace as the number of active users increases. Since the amount of workspace required by a user depends upon the nature of his activity, no simple formula can be given which will anticipate central memory requirements.

The file structure for RESPOND files stored on the system disk is designed to permit rapid access to individual lines of text. The SCOPE file system has as the basic allocatable item a half-track, which is 48 64-word sectors.** RESPOND breaks down each half-track assigned to it into 6 disk blocks of 8 sectors each and keeps a record of both the half-track number and block starting sector.

Each disk block, then, is 5,120 characters in length,

** Originally 50 sectors, the SCOPE half-track was reduced to 48 sectors in order to achieve a multiple of 8 sectors per half-track as required by RESPOND.

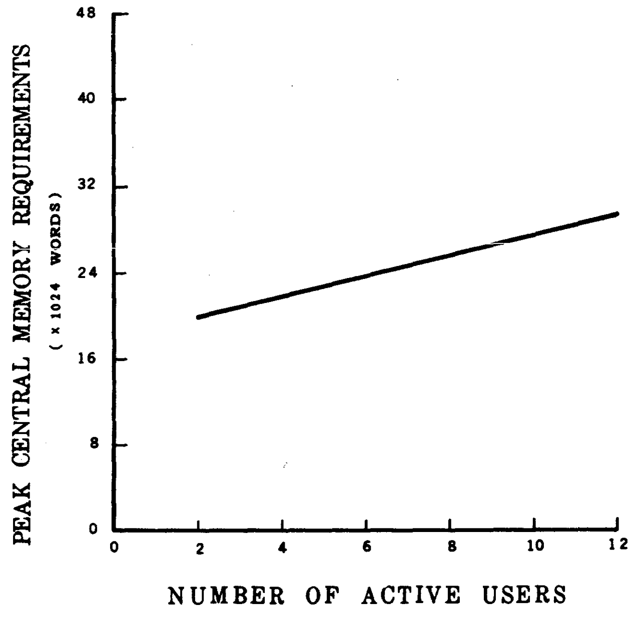


Figure 6—Peak central memory usage

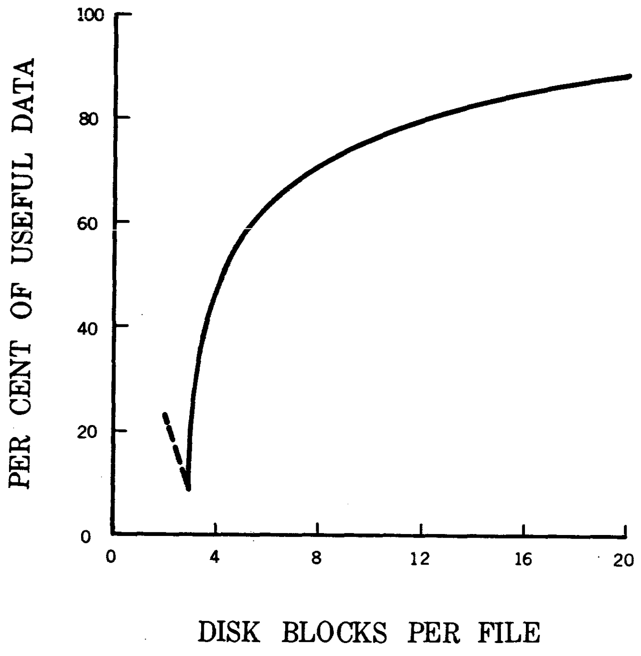


Figure 7—Disk file storage efficiency

and the file structure is such that any coded file requires a minimum of three disk blocks while binary files require a minimum of two disk blocks. Coded files are made up of one file header block and an arbitrary number of sequence number and text blocks. Since binary files have no sequence numbers associated with them, they do not require sequence number blocks. It has been observed that many RESPOND failures occur in the mapping operation between the SCOPE and RESPOND file structures described above.

Figures 7 and 8 show how user files are allocated on the disk by RESPOND. The portion of the file which contains useful data is plotted against the number of blocks required to contain that file. As the size of the file increases, the allocation efficiency of this file structure appears to stabilize near 85 percent. As Figure 8 indicates, however, the number of files which enjoy this allocation is a negligible percentage of the total number of user files. Since two-block files can only be binary files, the distributions shown in Figures 7 and 8 are slightly distorted. The data used in these figures represent 1,194 files distributed over 142 users. Ninety-one percent of the files were coded and nine percent were binary.

Aside from costs in terms of system resources, a remote console system can add other costs to the operation of a computation center. When such a system is first installed, it is very likely that several local modifications will be needed in order to tailor the operation

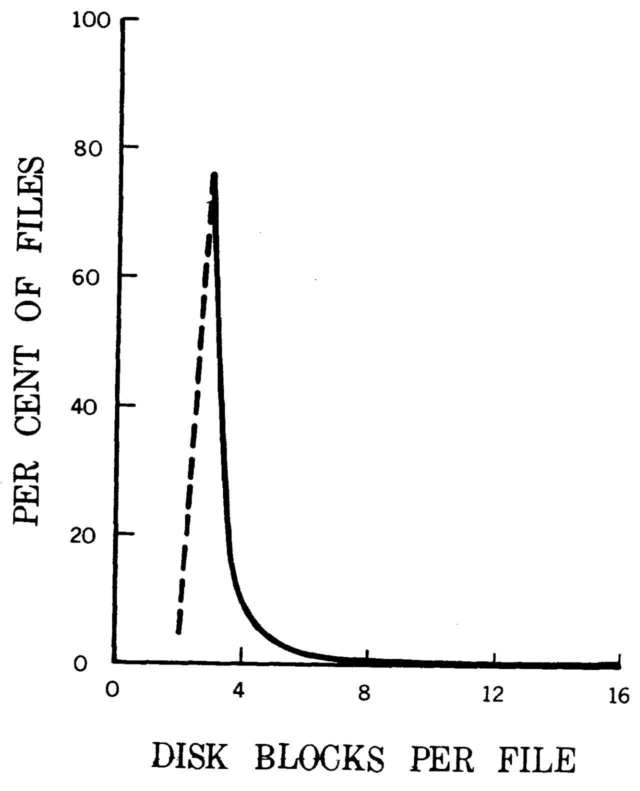


Figure 8—File size distribution

of the system to the particular user's environment. As usage of the system increases, demands for additional features, extended capability, and higher reliability will be voiced by users. Finally, like any other large, complex system, it is certain to have at least one remaining "bug" in it at all times.[7] For these reasons, the talent of a system programmer thoroughly familiar with the operation of the system will be required throughout the life of the installation.

A remote console system also demands the attention of the computer operators, a fact that is often overlooked in cost forecasts. It is estimated that at The University of Texas Computation Center, the computer operators spend about 1/10th of their time on tasks directly related to RESPOND. These tasks include replying to user messages on the master console, handling magnetic tape for some RESPOND jobs, loading and dumping RESPOND files, and intervening in the event of system failures. Since the remote console users are on-line, the computer operators must be prepared to service these demands in a punctual manner. They sometimes view this responsibility as a hindrance to their normal batch processing duties.

SUMMARY

A remote console system which has been in operation for more than 20 months has been analyzed. This analysis covered system performance, system reliability, user reaction, and cost. The study pointed out the following:

- System reliability is of paramount importance.
- A remote batch entry system such as RESPOND is very useful for many types of applications. Expanding its capabilities to include interactive processing and context-oriented text editing would make the system appeal to a larger class of users.
- It is extremely important that the remote console system and the operating system be compatible in as many areas (e.g., file structure) as possible. RESPOND's utilization of standard system software is considered a strong point in its favor.
- The cost of providing such a service is more than the expenditure of system resources such as processors, core memory, and mass storage. It also includes the talents of system programmers, computer operators, and a person to provide liaison between the remote users and the computation center.

REFERENCES

1 E A PEARSON
RESPOND, a user's manual
Computation Center The University of Texas Austin 1967

2 *TTY RESPOND reference manual*
Control Data Corporation Publication nr 60189300 1967

3 D F STEVENS
System evaluation on the Control Data 6600
Proc International Federation of Information Processing Societies CONGRESS 68 C34

4 A L SCHERR
Time-sharing measurement
Datamation Vol 12 No 4 22–26 April 1966

5 G E BRYAN
JOSS: 20,000 hours at the console—a statistical summary
Proc F J C C 1967

6 R V BUTLER
The Langley Research Center remote computing terminal system: implementation and first year's operation
Proc 21st National ACM Conference 1966

7 E W PULLEN D F SHUTTEE
MUSE: A tool for testing and debugging a multi-terminal programming system
Proc S J C C 1968

A model for core space allocation in a time-sharing system

by MAURICE V. WILKES

The University Mathematical Laboratory
Cambridge, England

INTRODUCTION

In a time-sharing system that is intended to serve a number of console users simultaneously, there are two related, but distinct, functions to be performed. One is *time slicing,* which is the allocation of bursts of processor time to the various active programs according to a suitable algorithm. The other is *core space allocation* which arises because, in a modern multi-programmed system, there will be space in core for more than one active program at the same time. If, as will normally be the case, there are more active programs than can be accommodated in core, some of them must be held on a drum and brought into core periodically; this is *swapping.* Confusion has sometimes arisen between time slicing and swapping, since, in the early time-sharing systems, there was only one active object program resident in core at any time, all the others being on the drum. In these circumstances, swapping and time slicing go together; when a program is in core, it is receiving processor time, and as soon as it ceases to receive processor time it is removed from core. In a multi-programmed system, however, space allocation and time slicing can proceed independently. It is the responsibility of the space allocation algorithm to ensure that, as far as possible, there is always at least one program in core that is ready to run. The time-slicing algorithm is responsible for dividing up the available processor time between the various programs that are in core.

Models can play a similar part in the discussion of computer systems as they can play in scientific theory. Practical situations tend to be highly complex, and a model serves the purpose of isolating and focusing attention on those features that are relevant to the purpose in hand. Since a model is simple, it can be defined precisely, and hence made to serve as a suitable basis for analysis, mathematical or otherwise. A model has essential features and non-essential features, and one object of analysis is to determine which are which. A model of a software system is not a blueprint for an implementation; a given model may, possibly, be implemented in a number of quite different ways. The implementer may depart from the model, for example, by adding features which spoil its simplicity but increase the running efficiency. In such cases, the analysed performance of the model gives a lower limit to the performance of the system. On the other hand, the implementer may be faced with practical limitations that the model maker could ignore, and these may lead him to a variety of compromises and sacrifices to expediency.

The model

This paper is concerned with core-space allocation for object programs only, and it is assumed that the supervisor is provided with a separate allocation system of its own. Practical reasons why this is desirable derive from the fact that a good deal of information is available about the likely behavior of supervisor processes. Some routines in the supervisor are needed so frequently that they must be kept permanently resident in core; in the case of others, it may be known that, when they have finished running, they will not be needed again for an appreciable time, and these processes, therefore, are best called down on each occasion when they are required. A system of *requesting* and *keeping* priorities specially adapted to the administration of the core space available for supervisor processes has been described by Hartley, Landy, and Needham;[1] this system lends itself to use in the case where, as in the model to be described, the amount of core space actually available to the supervisor varies dynamically from minute to minute. One of the responsibilities of the supervisor is handling input and output, and provision of the necessary space for buffers is dealt with by the space allocation procedure associated with the supervisor. There will be several further references to supervisor space scheduling, but the subject will not be discussed in detail.

In describing the model, it will be assumed that there is only one processor in the system. There is, however, no reason why there should not be more than one. The issue hardly affects the problem of space allocation, although it does, of course, affect the design of the time-slicing algorithm.

There is some difficulty in arriving at a nomenclature that is not ambiguous or misleading to describe the flow of work through a time-sharing system that is handling both a foreground and a background load. In the case of a batch-processing system, one can think of the work being presented as consisting of a series of self-contained jobs, each of which may pass through a number of stages variously known as job steps, phases, or tasks. If the same point of view is adopted in relation to a time-sharing system, then an on-line session at a console consists of a single job; a user, however, is more likely to think of himself as creating a number of separate jobs, some of which may run independently, and some run interactively. A distinction must also be made between programs that are run on behalf of a user, and programs that are run on behalf of the supervisor; the former are sometimes called object programs. This paper is concerned with the work in the system at a given time and not with the life history of individual jobs according to any particular definition of that term. The terms *job* and *object program* will, therefore, be used more or less interchangeably to refer to tasks or sub-tasks requiring to be done on behalf of a user and existing in the system at a given moment.

Swapping and resident regimes

Object programs that live on the drum and come in and out of core for periods of activation may be said to operate in the *swapping regime*; as pointed out above, such programs do not necessarily have the use of the processor for the whole time that they are in core. Programs that remain in core, and receive bursts of processor time at intervals, may be said to operate in the *resident regime*. In the present model, all programs, when first loaded, are operated in the resident regime, and those that survive for more than a short length of time pass into the swapping regime.

The explanation can be followed by reference to Figure 1. The lowest area of core is known as the *swapping area*, and is the area into which programs currently operating in the swapping regime are transferred in order to be eligible to receive processor time. In the simplest mode of operation, there is only one program

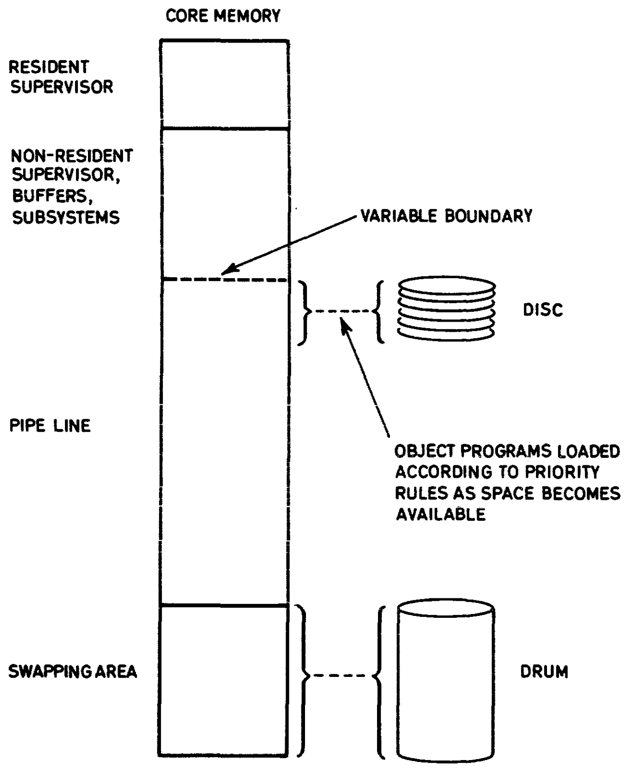

CORE MEMORY
RESIDENT SUPERVISOR
NON-RESIDENT SUPERVISOR, BUFFERS, SUBSYSTEMS
VARIABLE BOUNDARY
DISC
PIPE LINE
OBJECT PROGRAMS LOADED ACCORDING TO PRIORITY RULES AS SPACE BECOMES AVAILABLE
SWAPPING AREA
DRUM

Figure 1

at any time actually loaded into the swapping area, but more elaborate modes are possible. Above the swapping area comes the *pipeline*. This is large enough to contain several of the maximum-sized programs that are acceptable to the system. Programs enter the pipeline at the top and work their way down in the manner that will be described.* If they survive long enough, they eventually enter the swapping area. The top boundary of the pipeline is dynamically variable, and is indicated by a dotted line. Above this comes an area for holding sections of the supervisor that are kept on a drum or disc and brought into core only when they are needed. The same area is used to provide buffer space for the supervisor, space for lists maintained by the supervisor, and space for sub-systems that are subjected to supervisor-type scheduling, rather than to object-program type scheduling. Finally, at the top of the core, there is a region reserved for sections of the supervisor that are permanently held in core.

An object program starts its life by being queued on the disc, along with other object programs that have

* References to programs being moved down the pipeline do not necessarily imply that in an implementation programs should be physically moved in core. A system in which a similar effect were obtained by software devices or by paging hardware would be a valid implementation of the model.

been created either by a user or by the system. When space becomes available at the top of the pipeline, one of the waiting object programs is selected for loading by the *pipeline loading routine* in the supervisor. The pipeline loading routine is designed to favour short programs while ensuring that long ones are not indefinitely delayed, and, at the same time, to give effect to the differing priorities that may be attached to object programs, either as a result of administrative decision or by system requirements. When an object program enters the pipeline, it is given an upper limit for space (including space for data) that it may occupy, but no actual allocation of space is made at this moment beyond that immediately needed. Thus, a program of length 4K words that it is known will ultimately require another 8K for data is given an upper limit of 12K, but receives no more than 4K of physical space when first loaded into the pipeline.

When in the pipeline, object programs receive slices of processor time as determined by the time-slicing algorithm. Object programs that do not survive long enough to enter the swapping regime can leave the pipeline in various ways. They can become dead or dormant, the difference between these two states being that a dead program is needed no more and can be abandoned, whereas a dormant program must be transferred to the disc for possible reactivation, or post mortem examination, if required. An object program can also leave the pipeline as a result of going into a console wait; a discussion of what should happen in these circumstances will be reserved until later.

When space in the pipeline becomes free as a result of an object program leaving it, programs higher up in the pipeline are shifted down to fill the vacant space. This can result in space appearing at the top of the pipeline, in which case the pipeline loading routine can either load a new object program at once, or wait until additional space has become available. On the other hand, one of the other object programs in the pipeline, above the one that has disappeared, may be waiting for additional space, in which case the space now available (or as much as is needed) is given to this object program, any surplus being passed upwards for allocation to another waiting object program or to be made available to the pipeline-loading routine. One may think of a bubble of space passing upwards and either being absorbed or partly absorbed on the way, or reaching the top. If an object program survives long enough to reach the bottom of the pipeline, it is eligible to pass into the swapping area. The rate at which object programs are withdrawn from the pipeline controls ultimately how much space becomes available to the pipeline-loading routine and hence the rate at which new object programs are loaded.

The efficient operation of the system requires that the pipeline shall contain a sufficient number of object programs, or jobs as they will now usually be called; there is no point, however, in increasing the number of jobs beyond the point at which there is a high probability that, at any instant, there will be at least one job in core that is ready to run. The criterion here is the number of jobs in the pipeline, not the total core space that they occupy. The amount of space needed in the pipeline will, therefore, vary from second to second, according to the average size of the jobs. When there is space to spare, it is better to give it (temporarily) to the supervisor rather than to load redundant jobs. The supervisor will be able to make good use of the space for purposes already mentioned, such as holding non-resident routines and temporary buffers. In the model, therefore, we assume that the pipeline loading algorithm is designed to keep the number of jobs in the pipeline constant.

The pipeline has the effect—and this indeed is its purpose—of filtering out jobs that run for short periods only, since these will leave the pipeline before they reach the bottom. Since they never reach the swapping area, the overheads of swapping are avoided altogether in the case of such jobs.

Jobs that are interacting closely with a console will typically run for short periods, and go into frequent console waits. Such jobs are, therefore, most suitably dealt with by the pipeline technique, rather than by the swapping technique, and, of the various strategies available, the following would appear to be the best calculated to give a high standard service to such jobs without interfering with the smooth running of the system as a whole.

A highly interactive job enters the pipeline in the manner that has been described with no special privileges; it will not, in fact, be known to the system at this time that the job is highly interactive. The job may be expected to reach a console wait while it is still in the pipeline, although there is no reason why it should not run long enough to enter the swapping area. The essence of the strategy proposed is that, when reactivated by a response from the console, the job should re-enter the pipeline at the top. A job reaching a console wait is, therefore, immediately removed from the pipeline and, on reactivation, is returned to the queue of jobs waiting to enter. In order, however, that it should not be subject to the delays that normally occur at this point, it is handled according to special rules; what these are depend on the importance attached to giving the best possible response to jobs while they are interacting with a console. An advantage of the model is that it enables one to see what is the cost of the various steps that can be taken to this end, both in terms of the employment of system resources and the effect on the throughput of other types of job. One step that will naturally be taken is to provide that a console wait job shall be placed on a separate queue and given priority over jobs waiting on the regular queue to enter the pipeline. Further improvement can be obtained by implementing this special queue on a drum instead of on the disc; in the model, the drum used for this purpose would be shown as separate from the drum used for swapping, although, of course, in an implementation, the same physical drum might be used for the two purposes. A further step that could be taken to improve the response after a console wait would be to design the pipeline loading algorithm so as to keep in hand a certain amount of space at the top of the pipeline for the accommodation of console-wait jobs when they are reactivated. The more space kept in hand, the better will be the service given to highly interactive jobs, but the greater will be the cost to the system. The balance to be struck is a matter for management decision in any particular case.

Since an interactive job returns to the queue of jobs waiting to enter the pipeline when it comes to a console wait, it is treated by the system as though it were a new job, although one having special priority. In what follows, therefore, the term 'job' will for brevity be used to denote either an entirely new job or an interactive job that has been resuscitated after a console wait.

Jobs may be held up waiting for other forms of peripheral action, such as disc or magnetic-tape transfers. The former are of brief duration, and the jobs concerned continue their progress through the system. Magnetic tape waits, however, are of relatively long and uncertain duration; they have a good deal in common with console waits, and it can be argued that they should be dealt with in a similar way.

Time-slicing algorithm

The core space allocation system ensures that, at any given time, there are sufficient object programs in core to make effective multiprogramming possible, and that any programs resident on the drum come into core often enough to be able to receive processor time at appropriate intervals. The time-slicing algorithm determines how the processor time is allocated to the various programs that happen to be in core and are free to run, whether they are in the pipeline or in the swapping area. It may be assumed, although this is not strictly necessary, that time is shared between the pipeline and the swapping area in a fixed proportion. Time is given to programs in the pipeline in small slices; the smaller the slice the better, provided that program changing overheads do not account for a significant fraction of the total time. Note that a program temporarily resident in the swapping area will not be active during the whole time that it is in core. Like programs in the pipeline, it will receive time in small slices; it will be returned to the drum when the total amount of time that it has received since being loaded reaches a certain figure, chosen to be high enough to make the overheads of swapping worthwhile. This figure would be greater for long programs than for short ones since the swapping time depends on a program's length.

Swapping regime

If the allocation of processor time to jobs in the swapping regime is a fixed proportion of the total time available, then the rate at which jobs terminate depends only on their average expectation of life at the moment they enter the swapping region; in fact, the rate at which jobs terminate is $1/\bar{\mu}$, where $\bar{\mu}$ is their expectation of life, calculated on the assumption that there would only be one job in the swapping regime. The rate of termination of jobs is, in particular, independent of the number of jobs in the swapping regime. Thus, the swapping regime may be likened to a hopper from which objects are extracted at a given mean rate. Since the rate of entry and the rate of abstraction are subject to stochastic variations, the number of jobs in the swapping regime will fluctuate about a mean, even if the mean rate of abstraction is equal to the mean rate of entry. The problem of investigating these variations may be tackled by the methods of queuing theory.

The simplest method of core space allocation for the swapping regime is to bring one program into core at a time; the time-slicing algorithm has then only one program in the swapping area to be concerned with. Almost as simple, if the hardware permits, is the *wrap around* method in which the section of core constituting the swapping area is addressed *modulo* n, where n (in practice a power of) is the number of words it contains. Each new program brought down starts where the previous one left off and if the programs are short several can be in core at once. A maximum-sized program will, of course, occupy the whole swapping area. The time-slicing algorithm can be designed to take advantage of the fact that there may be more than one program in the swapping area at a given time. The result is an improvement in the average efficiency of the multiprogramming.

Once a program has entered the swapping regime, its troubles as regards acquiring core space are over, and it can occupy as much core (up to the maximum permitted to any program) as it requires.

The pipeline

If the number of jobs in the pipeline is held constant, then the methods of standard queuing theory are not applicable. An approximate treatment is, however, offered in the Appendix. For a given rate of abstraction of jobs from the pipeline this enables the distribution of age of those jobs to be computed, and also the rate at which jobs must be loaded to keep the pipeline full.

In order to carry through the calculations, it is necessary to assume a form for the statistical distribution of job life. In one particular case (the Poisson case) the treatment becomes exact. This is the case in which the expectation of life of a job is independent of its age.

The practical effect of the pipeline is to filter out short jobs before they can enter the swapping regime. It is of interest to know the expectation of life of jobs emerging from the pipeline since this determines their behavior in the swapping regime, namely, how long they are likely to remain in it, and how many times they are likely to be swapped. In the Poisson case, the probability of a job reaching the end of its life in an interval δt is independent of its age, and its expectation of further life is also independent of its age. In this case, the expectation of life of jobs entering the swapping area will be equal to their expectation of life when they enter the pipeline. If, however, the probability of a job reaching the end of its life in δt increases with age, so that a job that has survived an initial period is unlikely to continue for a long time, then the expectation of life of jobs entering the swapping area will be less than their initial expectation of life. If this is so to any marked degree, the effectiveness of swapping is open to question, since very few jobs will survive more than one swap, and an increase in the length of the pipeline sufficient to allow all jobs to finish while still within it would be more suitable.

Implementation

It has already been mentioned that systems in which programs are not physically shifted in core, but in which similar effects are achieved by other means, are to be regarded as valid implementations of the model here described; the model is, in fact, much more general than the description given above may at first sight suggest. Essential features, however, are (1) that when a job is first loaded it is given a period of continuous residence in core before a regime in which regular swapping to and from a drum is initiated, and (2) that space is given to a program piecemeal as it needs it and not all at once. These objectives can be achieved by a direct implementation of the shifting described in the model or by making use of a hardware paging system. In relation to the latter case, the discussion given here is really a discussion of the way in which the paging algorithm should be designed. The objectives can also be achieved approximately if programs on first loading are so located in core that enough contiguous space to meet their ultimate needs can be earmarked for their use. Until such time as they need the space, it can be made available to the supervisor for accommodating non-resident routines and for buffering.[1]

Overall control of the system

One of the objects of establishing a model for core space allocation is to enable the control problem for the system as a whole to be formulated. The problem of controlling a time-sharing system has much in common with control problems met in the process industries, and this fact will be brought out by describing a closely analogous problem connected with the control of an ore-grading plant.

In an industrial plant, there are commonly a number of purely local control loops presided over by controllers that operate independently of the main control system. One such control loop—connected with the balance between foreground and background jobs—can be identified in the time-sharing system under discussion. Jobs that are ready to enter the pipeline wait on one of a number of queues on the disc. In the simplest case there will be separate queues for foreground jobs and for background jobs, the latter including background jobs initiated from consoles. There is, in addition, a queue for jobs waiting to be reloaded after a console wait. Jobs are loaded into the pipeline according to rules designed to give priority to jobs waiting to be reloaded, and otherwise to favor the foreground to the extent determined by operational requirements. These rules can be designed in such a way that minor short-term variations in the foreground load can be accommodated by varying the rate at which background jobs are fed, thus avoiding the need for significant variation in the rate of flow of jobs into the pipeline. This is a piece of local control of a straightforward kind. Longer term changes—up or down—in the foreground load re-

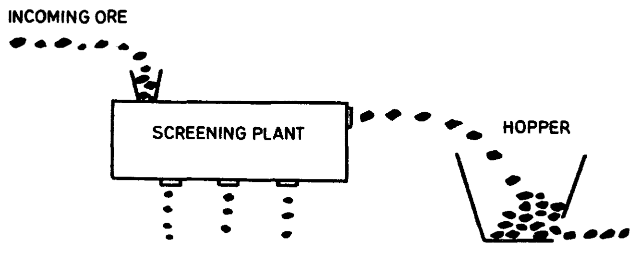


Figure 2

main to be dealt with by the overall control mechanism, which will forcibly log out a proportion of the users (after giving them a warning) when the load is heavy and allow extra users to log in when the load is light.

The closely analogous problem that will be considered is that of controlling the ore-grading plant illustrated in Figure 2. Lumps of ore of varying sizes enter a screening plant and the smaller ones fall out. The larger lumps continue and pass into a hopper from which they are extracted at a constant rate by a conveyor. The screening plant corresponds to the pipeline and the hopper to the swapping area. Ore is fed to the plant from an external source, and the rate of flow can be controlled, although response to control signals is not rapid. This corresponds to adjusting the number of console users of a time-sharing system in the manner just described.

The input parameters on which control of the ore-grading plant must be based are (1) a measurement of the amount of the material in the hopper, and (2) a measurement of the amount of material that has piled up at the entry to the screening plant. There must be two output signals from the control system; of these, one is used to control the rate of flow from the screening plants to the hopper, and the other is sent to the external source of supply and used to control the rate of feed of ore into the plant. The design objectives of the control system are, in the short-term, to make use of the storage capacity available in the hopper to prevent any appreciable piling up of material at the entry to the screening plant and, in the long-term, to adjust the rate of arrival of material so that the system operates smoothly and efficiently with as little material as possible in the hopper.

Time delays are a common cause of instability in the operation of a plant if the control system is not carefully designed. In the time-sharing system, instability could occur on account of the fact that changes in load—that is, changes in the number of users logged in—cannot be made instantaneously. If, for example, the control mechanism, faced with an increase of activity on the part of the users currently logged in, were to overestimate the number that must be warned off, the system would, at some later time, be found to be underloaded. Over-correction of this situation would, in turn, give rise to eventual overloading, and the system would proceed to oscillate between one extreme and the other. On the other hand, the use of a control mechanism that achieved stability, and avoided overloading, by being unduly cautious in allowing the number of on-line users to grow when the system was underloaded would obviously result in the system running below capacity most of the time as far as service to on-line users was concerned. These are typical problems encountered in control engineering, and it is suggested that the designers of time-sharing systems could learn something from their colleagues working in that discipline.

REFERENCE

1 D F HARTLEY B LANDY R M NEEDHAM
The structure of a multiprogramming supervisor
The Computer Journal 11 No 3 p 247 November 1968

APPENDIX

Analysis of pipeline

The following approximate analysis applies to the case in which the number of jobs emerging from the pipeline is small compared with the number of jobs entering.

Let $P(\ell)$ be the probability that a job entering the pipeline has a life* greater than or equal to ℓ and let

$$p(\ell) = -\partial P(\ell)/\partial \ell \quad ; \quad p(\ell)\,\delta\ell \quad \text{is}$$

the probability that the job finishes in the interval $\delta\ell$, i.e., between ℓ and $\ell + \delta\ell$. The expectation of life of a job entering the pipeline is then

$$\bar{\ell} = \int_0^\infty t\, p(t)\, dt$$

By integrating by parts it may be shown that $\bar{\ell}$ is also given by

$$\bar{\ell} = \int_0^\infty P(t)\, dt \qquad (i)$$

Consider a pipeline containing n jobs from which no withdrawals are made, but in which each job is replaced by a new one as soon as it finishes. The probability of a job selected at random at a randomly chosen instant having an age greater than or equal to ℓ is then

$$A(\ell) = (1/\bar{\ell}) \int_\ell^\infty P(t)\, dt$$

The probability that the oldest of the n jobs existing in

* For this purpose, a job reaches the end of its life when it becomes dead or dormant, or reaches a console wait. In this Appendix, time is true elapsed time and the distributions take account of the fact that processor time is being shared among a number of jobs.

the pipeline at a randomly chosen instant has an age in $\delta\ell$ is $Q_n(\ell)\ \delta\ell$ where

$$Q_n(\ell) = (1/\bar{\ell}) \frac{\partial}{\partial\ell} [1 - A(\ell)]^n$$

In practice, a job selected as being the oldest of the n jobs in the pipeline at a random instant is withdrawn and replaced by a new one. If the rate of withdrawal is small compared with the natural death rate of jobs in the pipeline, it may be assumed as an approximation that $Q_n(\ell)$ still gives the age distribution.

Let the extra life that a job withdrawn at age ℓ would have had if it had remained in the pipeline be μ. Then the expectation of total life $\ell + \mu$ is given by

$$E(\ell + \mu) = \int_{\ell}^{\infty} t\ [p(t)/P(\ell)]\ dt$$

$$= -\ [1/P(\ell)] \int_{\ell}^{\infty} t\ [\partial P(t)/\partial t]\ dt$$

$$= \ell + \bar{\ell}\, A(\ell)/P(\ell)$$

on integrating by part and using (1). Thus $E(\mu) = \bar{\ell}\ A(\ell)/P(\ell)$

If this value for $E(\mu)$ is averaged over the distribution $Q_n(\ell)$, we have the expectation, L, of the amount by which the life of a job in the pipeline is shortened by being withdrawn:

$$L = \bar{\ell} \int_{0}^{\infty} [Q_n(\ell)\ A(\ell)/P(\ell)]\ d\ell$$

It is now possible to arrive at the following relationship between the number, N, of jobs entering the pipeline during a period T_0 and the number, W, withdrawn during the same period:

$$N\bar{\ell} - WL = nT_0$$

or

$$N = (nT_0 + WL)/\bar{\ell}$$

If $P(\ell) = \exp(-\alpha\ell)$ (the Poisson case) then it is well known that the expectation of life is independent of age. In the above notation, as may easily be verified,

$$E(\mu) = \bar{\ell}$$

The theory then becomes exact, and we have

$$N = nT_0/\bar{\ell} + W$$

The number of jobs that finish in the pipeline is independent of the withdrawal rate, and if more jobs are taken out then a similar number of extra jobs must be put in. The expected life of a job on emergence is the same as its expected life on entry.

It may be observed that no job can remain in the pipeline for more than n sampling intervals. The approximation given by $Q_n(\ell)$ to the age distribution of jobs in the pipeline, subject to withdrawal, may be improved by redefining $A(\ell)$ as follows:

$$A(\ell) = (1/\bar{\ell}) \int_{\ell}^{n\tau} P(t)\ dt$$

where τ is the average interval between withdrawals. The results of a series of simulations suggest that, with this refinement, the theory is sufficiently precise for most practical purposes.

Picture-driven animation*

by RONALD M. BAECKER**

*National Institutes of Health****
Bethesda, Maryland

INTRODUCTION

"Animation is the graphic art which occurs in time. Whereas a static image (such as a Picasso or a complex graph) may convey complex information through a single picture, animation conveys equivalently complex information through a sequence of images seen in time. It is characteristic of this medium, as opposed to static imagery, that the actual graphical information at any given instant is relatively slight. The source of information for the viewer of animation is implicit in picture change: change in relative position, shape, and dynamics. Therefore, a computer is ideally suited to making animation "possible" through the fluid refinement of these changes."[27]

The animation industry is ripe for a revolution. Historical accidents of available technology and knowledge of visual physiology have led to the evolution of the animated film as "one that is created frame-by-frame."[1] The prodigious quantities of labor required for the construction of twenty-four individual frames per second of film have led to a concentration of animation activity in the assembly-line environments of a few large companies, an artificial yet rarely surmountable separation of the artist from the medium, and extravagant costs.[2] In conjunction with other trends in American society, the result is usually what the English critic Stephenson describes as "the respectable sadism and stereotype of commerce."[1] Yet he offers this hopeful prediction in concluding his 1967 study, *Animation in the Cinema*: There seems every reason to look forward to changes which would make it possible for the creative artist to put on the screen a stream of images with the same facility as he can now produce a single still picture."[1] This paper explains how a creative artist, aided by a computer, can define a stream of images with the same facility as he can now produce a very few still pictures.

Although the computer's entrance into animation has been a recent one (1964),[3–4] the growth of interest and activity has been phenomenal.[5–8] Experience to date strongly suggests that the following statements are true:

1. The animated display is a natural medium for the recording and analysis of computer output from simulations and data reduction, and for the modeling, presentation, and elucidation of phenomena of physics, biology, and engineering.[9–15] Depiction through animation is particularly appropriate where simultaneous actions in some system must be represented. If the animation is the pictorial simulation of a complex, mathematically-expressed physical theory, then the film can only be made with the aid of a computer.
2. The computer is *an artistic and animation medium*, a powerful aid in the creation of beautiful visual phenomena, and not merely a tool for the drafting of regular or repetitive pictures.[16–19]
3. The formal modeling of pictures by complexes

* Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract NONR-4102(01), and by M.I.T. Lincoln Laboratory with support from the U.S. Advanced Research Projects Agency.

** This paper is based on a thesis submitted in partial fulfillment for the degree of Doctor of Philosophy at the Massachusetts Institute of Technology, Department of Electrical Engineering.

*** Division of Computer Research and Technology.

of algorithms and data facilitates the continued modification of a single animation sequence and the production of a series of related sequences.

This paper discusses ways in which man, aided by a computer in an *interactive graphical* environment, can synthesize animated visual displays. It is widely recognized that such an environment facilitates man-machine communication about still pictures.[20–22] The paper seeks to:

1. describe the role of direct graphical interaction and sketching in computer animation, resulting in the process we shall call *interactive computer-mediated animation*; and,
2. develop a new approach to the specification of picture dynamics, one which exploits the capacity for direct graphical interaction. The result we shall call *picture-driven animation.*

Animation in an interactive computer graphics environment

The role of direct graphical interaction in the synthesis of animated visual displays

Three aspects of the role of direct graphical interaction in computer graphics are particularly relevant to computer animation:

1. The availability of immediate visual feedback of results, final or intermediate;
2. The ability to factor picture construction into stages, and to view the results after each stage; and,
3. The ability to sketch pictures directly into the computer.

The power of immediate visual feedback in animation is striking. The computer calculates, from its representation of a dynamic sequence, the individual frames of the corresponding "movie." Like a video tape recorder, it plays it back for direct evaluation. A small change may be made, the sequence recalculated, and the result viewed again. The cycle of designation of commands and sketching by the animator, followed by calculation and playback by the computer, is repeated until a suitable result is achieved. The time to go once around the feedback loop is reduced to a few seconds or minutes. In most traditional and computer animation environments, the time is a few hours or days. The difference is significant, for now *the animator can see and not merely imagine the result of varying the movement and the rhythm of a dynamic display.* Thus he will be led to perfect that aspect of animation that is its core: control of the changing spatial and temporal relationships of graphic information.

Factoring the construction of an animation sequence facilitates the effective use of feedback from early stages to guide work in later stages. Working on individual small subsequences helps overcome the serious practical problems of computer time and space that could disallow rapid enough calculation and playback.

We know from the computer graphics of still pictures that the computer simulates not only a passive recording agent in its ability to retain images, but an active medium which transforms the very nature of the sketching process. This remark applies trivially to computer animation; one may construct a sequence of drawings to comprise the individual frames of the film, the static images existing at single instants of time. Picture change that extends over entire intervals of time is then synthesized as a succession of individual (temporally) *local* changes that alter one frame into another.

This paper goes further, for it explains how the computer can be a medium which transforms the very nature of the process of defining picture *change*, of defining movement and rhythm. Dynamic behavior is abstracted by *descriptions of extended picture change.* These descriptions may themselves be represented, synthesized, and manipulated through pictures, both static and dynamic. Thus dynamic control can be exercised *globally* over the entire sequence. What results is one new conception of what it means to draw an animated film.

The components required to realize an interactive computer-mediated animation system

Interactive computer-mediated animation is the process of constructing animated visual displays using a system containing, in one form or another, at least the following eight components:

Hardware:

1. A general-purpose digital computer.
2. A hierarchy of auxiliary storage. This is listed separately to emphasize the magnitude of storage required for the data structures from which an animation sequence is derived and for the visual images of which it is composed.
3. An input device such as a light pen, tablet plus stylus,[23–24] or wand,[25] which allows direct drawing to the computer in at least two spatial dimensions. The operating environment must, upon user demand, provide at least brief intervals during which the sketch may be made in real time. The animator must then be able to

draw a picture without any interruption. Furthermore, the computer must record the "essential temporal information" from the act of sketching. Sampling the state of the stylus 24 times per second often suffices for our purposes.

4. An output device, such as a standard computer display scope or a suitably modified TV monitor, which allows the direct viewing of animated displays at a rate such as 24 frames per second. This is essential to enable the interactive editing of animation subsequences. The final transmission of a "movie" to the medium of photographic film or video tape can but need not use the same mechanisms.

Software:

5. A "language" for the construction and manipulation of static pictures.
6. A "language" for the representation and specification of picture change and the dynamics of picture change. We shall introduce in this paper methods of specifying dynamics not possible with traditional animation media and not yet attempted in the brief history of computer animation.
7. A set of programs that transforms the specifications of picture structure and picture dynamics into a sequence of visual images.
8. A set of programs that stores into and retrieves from auxiliary memory this sequence of visual images, and facilitates both its real time playback for immediate viewing and its transmission to and from permanent recording media.

Figure 1 portrays a suitable environment for interactive computer-mediated animation. Figure 2 is a block diagram of such a system.

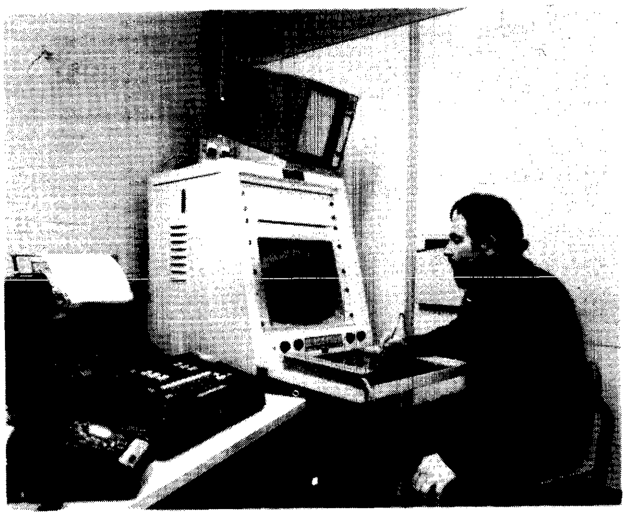


Figure 1—An interactive computer-mediated animation console. The author is sketching with the stylus on the tablet. There is a CRT for viewing dynamic displays, a storage scope above it, a typewriter, knobs, toggle switches, and a telephone so that an animator may summon help

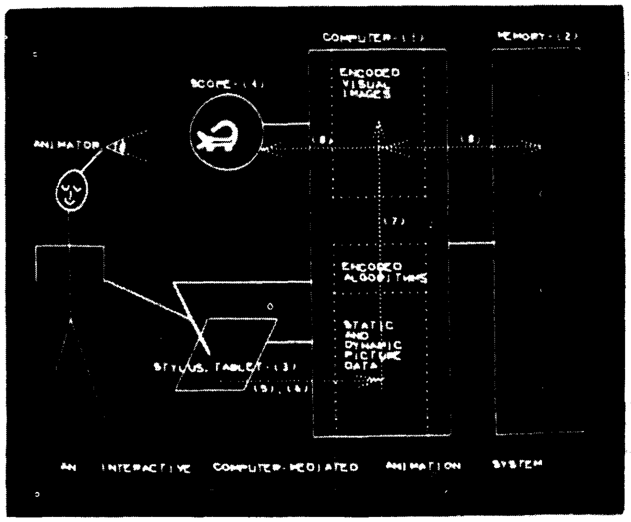


Figure 2—Block diagram of a minimal system for interactive computer-mediated animation. The parenthesized numbers refer to the system components defined in the paper.

A scenario illustrating the use of an interactive computer-mediated animation system

To illustrate the process of animation in an interactive computer graphics environment, we present a scenario. The example, chosen for its simplicity, is an extended version of one actually executed with the GENEralized-cel animation SYStem. GENESYS is a picture-driven animation system implemented on the M.I.T. Lincoln Laboratory TX-2 computer. All capabilities purported to it are operational or could be made so by minor additions. The written form of the interactive dialogue has been adjusted to increase its clarity.

We want to see a dynamic sequence of a dog dashing to his dinner and then dining: The dog runs towards a bowl. Wagging his tail, he lowers his head and laps up the milk. Several slurps of the milk are to be shown before we cut to the next scene.

How we do it:

ANIMATOR(A): *CALL* GENESYS;
GENESYS(G): *HELLO. GENESYS AWAITS YOUR CREATION*;

GENESYS either types or displays this response.

A: *FORMMOVIE* DINNERTIME;
The animator either types the command name '*FORMMOVIE*', hits a corresponding light-button with the stylus, or writes an abbreviation of the command name to a character-recognizer.[26] He then types a movie name, 'DINNERTIME'.

G: *FRESH*;
No such movie exists in the animator's directory. Hence, work begins on a totally new one.

A: *FORMBACKGROUND*;
A. wants to define a subpicture that will be visible in all frames of the sequence.

G: *SKETCH IT, MAN*;

A: . . . . .
A. sketches the bowl, drawing with the stylus on the tablet. What he draws appears immediately on the display scope.

G: *OK*;

A: *FORMCEL* #1 *in CLASS BODY*;
An initial version of the dog's body is to be made a unique subpicture, or cel.
. . . . .
He sketches it, and soon adds one version of the legs, tail, and head, each as a unique cel in a unique cel class. Now, a coherent dog, unmoving, appears on the scope.
. . . . .

A: *BIND* BODY, LEGS, TAIL, HEAD, TONGUE;
This guarantees that any translational motion applied to the dog will drive the body, legs, tail, head, and tongue together. Thus the dog won't disintegrate while moving.

G: *OK*;

A: *SKETCHPCURVE* BODY;
. . . . .
A. now sketches the path of the desired motion, mimicking the movement with the action of his stylus. Hop . . . hop . . . hophophop . . . goes his hand. The act of mimicking a continuous movement is called a p-curve.
. . . . .

A: *PLAYBACK*;
Playback the current version of the movie. Hop . . . hop . . . hophophop . . . glides the rigid dog across the scope towards the bowl. Four frames from such a motion are shown superimposed in Figure 3.
. . . . .

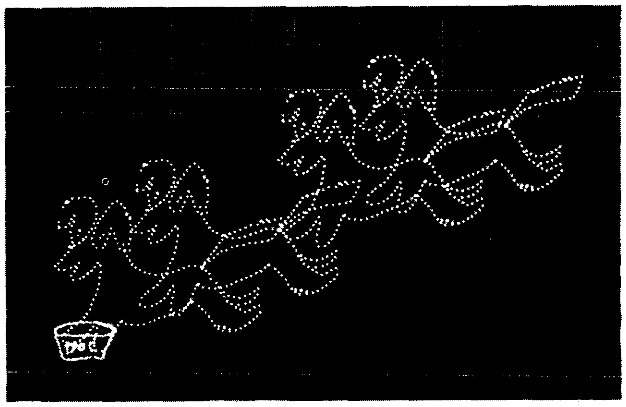


Figure 3—A static dog glides towards a bowl. The sketches are by Mrs. Nancy Johnson of Waltham, Massachusetts

A: *FORMCEL* #2 *in CLASS* LEGS:
. . . . .
A. sketches the legs in another position, that is, he defines the second cel in the class 'LEGS.' This may be followed by several more positions. The images are ones that are useful in synthesizing running and hopping movements.
. . . . .

A: *TYPESELECTIONS from* LEGS;
. . . . .
He types in a sequence of choices of one of the positions of the legs. Each succeeding choice selects which cel is to be displayed as the dog's legs in the next frame. Of course only one set of legs is visible in a frame.
. . . . .

A: *PLAYBACK*;
Now, as is portrayed in Figure 4, the legs move

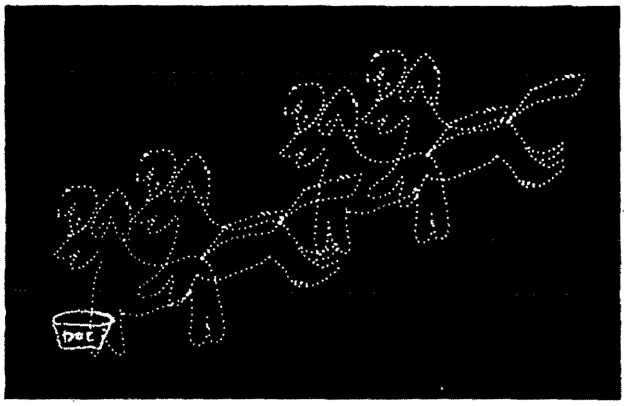


Figure 4—Now the dog hops to the bowl

while the dog hops to the bowl.

. . . . .

Further refinements to the leg motion are made. This includes the resketching of one cel. The tail and head movements are similarly introduced. The sequence then appears as is shown in Figure 5.

. . . . .

Three tongue cels are sketched.

. . . . .

A: *TYPESELECTIONS from* TONGUE;

. . . . .

For most of the sequence, the zeroth tongue is selected, that is, no tongue is visible. A single lap, or slurp of the tongue is synthesized from the three tongue positions, and is introduced at the appropriate time in the movie. The leftmost image of Figure 6 shows the extended tongue.

. . . . .

A: *TAPRHYTHM* SLURPINTERVALS;

. . . . .

A. can feel or intuit the rhythm of the desired slurps

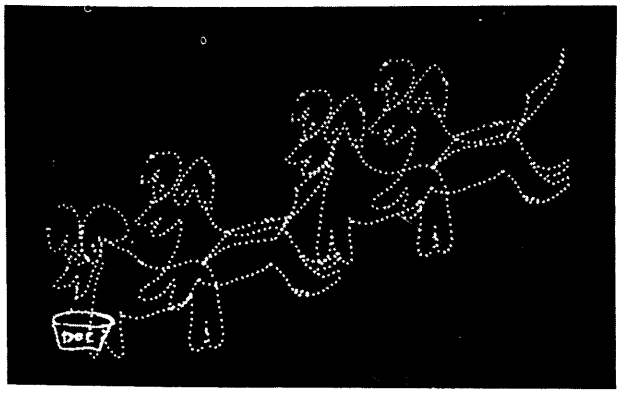


Figure 5—Eager for dinner, he wags his tail

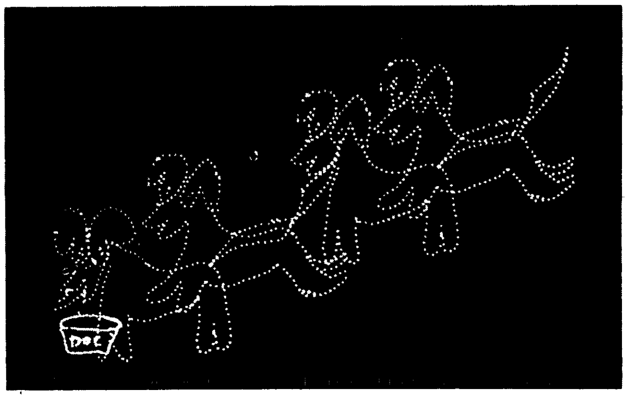


Figure 6—Slurp goes his tongue, lapping up the milk

better than he can rationalize it. Hence he goes tap . . . tap . . . taptap . . . on a push-button.

. . . . .

A: *REPEATPATTERN FROM frame* 59 *THROUGH frame* 64 *of SELECTIONS from* TONGUE *atINTERVALS of* SLURPINTERVALS;

Assume that the visual slurp occurs in frames 59 through 64. The pattern of tongue selections which yields the slurp is repeated at intervals determined by the tapped rhythm.

. . . . .

A: *PLAYBACK*;

Now the dog goes hop . . . hop . . . hophophop . . . slurp . . . slurp . . . . slurpslurp.

. . . . .

The movie is essentially complete; minor refinements may now be made.

. . . . .

A: *EDIT X WAVEFORM of* BODY;

. . . . .

More acceleration in the hopping movement would better portray the dog's eagerness for his dinner. Hence, A. calls forth a display of the dog's X coordinate versus time, and resketches part of the waveform so that there is more horizontal acceleration.

. . . . .

A: *EDIT FRAME* 44;

. . . . .

Assume that the dog reaches the bowl in frame 44. Viewing the sequence in slow motion, A. notices that the dog's position at the bowl could be improved. He alters its location in frame 44 using the knobs under the scope.

. . . . .

A: *FIX X and Y of* BODY *AFTER frame* 44;

The path descriptions are further modified so that the dog again holds a fixed position, once it has reached the bowl.

. . . . .

A: *PLAYBACK*;

. . . . .

A: *SAVE* DINNERTIME;

The movie is saved, available for further refinements at any time.

G: DINNERTIME *IS SAVED. GOOD BYE.*

Implications of the scenario

1. Approximately 100 frames have been generated

from fewer than 20 cels. Only very limited tools have been used in cel construction, specifically, programs that accept direct sketches and that enable selective erasure of picture parts. Nonetheless, great power results from the animator's ability to control and evaluate dynamic combinations of a few static images.
2. Immediate playback enables interactive experimentation to achieve desired visual effects. The actions described above, including considerable trial-and-error, may be completed in well under one hour, even if all cels must be constructed anew.
3. A variety of static images, analytical graphs of picture action, depict the time dependence of dynamic picture parameters. An example is the waveform representing the dog's changing horizontal position. Viewing such static representations aids the understanding of existing animation sequences; resketching or editing them changes the actual dynamic behavior accordingly.
4. The animator may in real time mimic aspects of dynamic behavior. His movement and rhythm are recorded by the system for application in the movie. This occurs when the hopping of his stylus motion is used to drive the dog, and when the tapping of a push-button is used to determine the rhythm of the slurps of the tongue.
5. Three aspects of dynamic behavior appear in the example: *path descriptions*, or conceptually continuous coordinate changes; *selection descriptions*, or recurring choices of cels from a cel class; and *rhythm descriptions*, or temporal patterns marking events. The pictures (3) and actions (4), through which direct control over dynamics is exercised, are representations of these three kinds of *global descriptions of dynamics*.
6. Global operations (3)-(4), which alter dynamic behavior over entire intervals of time, may be supplemented where necessary by local operations, which adjust individual frames. An example is the positioning of the dog near the bowl.

The specification of picture dynamics

Three old approaches to the definition of picture dynamics

We may distinguish three old approaches to the synthesis of a sequence of frames:

1. The individual construction of each frame in the sequence;
2. The interpolation of sequences of frames intermediate to pairs of critical frames; and,
3. The generation of frames from an algorithmic description of the sequence.

Animation sequences have traditionally been synthesized through the individual construction of frames. The illusion of a continuum of time is attained through rapid playback of discrete instants of time. This approach is the only one applicable to the construction of pictures that defy regular or formal description, and that require unique operations on each frame. Yet the cost is excessive and continues to rise dramatically, faster than the GNP.[27] Salaries in large studio operations typically consume half of the cost, for commercial animation is a complex interaction among producers, directors, designers, layout artists, background artists, key animators, assistant animators, inkers and colourists, checkers, cameramen, editors, and studio managers.[2] It is this division of labor, this dispersal of the creative process, which separates the artist from the medium.[27] Another major weakness of conventional frame-by-frame animation is that there are no efficient methods of making changes to a movie stored on photographic film or video tape. We discuss elsewhere what role the computer might assume in frame-by-frame animation.[28]

The technique of interpolation has long been used to cut costs and reduce the burden of picture construction which is placed on the key animator. Interpolation occurs when the key animator asks his assistants to fill in the pictures intermediate to a pair of critical frames. It has been suggested that part of this process could be mechanized.[29] We do not consider further that problem in this paper.

The generation of a sequence of frames from a formal algorithmic description is a process characterized by:

1. the need to use a computer, for it is the only animation medium which can follow and execute with ease a complex algorithm;
2. generality, that is, applicability to a large class of regularly-structured pictures;
3. representational power, or the compactness with which interesting animated displays may be formulated; and,
4. flexibility and adaptability, or the ease with which a variety of alterations may be made to a movie expressed as an algorithm.

The form of the expression has to this date been a written program in a picture-processing language such as BEFLIX,[3-4] or a sequence of directives in a typewriter-controlled command language such as CAFE.[30] Herein lies another strength of the approach and also a fundamental weakness. On the one hand, many programmers, scientists, and engineers, previously not animators but fluent in this new "language," can now produce dynamic displays.[31] On the other hand, an animator trained in traditional media and techniques is forced to learn a completely new "language," a completely new way of thinking.

One new approach to the definition of dynamics —picture-driven animation

Picture-driven animation is a new process that *augments* harmoniously the animator's traditional techniques, that reflects and *extends* the ways of thinking to which he is accustomed. Within his intuitive "language" of pictures and sketching and mimicking, he may synthesize both components of frames, called *cels*, and generative descriptions of extended picture change, called *global descriptions of dynamics.*

Global dynamic descriptions are data sequences, whose successive elements determine critical parameters in successive frames of the movie. Algorithms embedded in a picture-driven animation system combine cels and dynamic descriptions to produce visible picture change. The animator defines and refines pictorial representations of dynamic descriptions. These data sequences then "drive" the algorithms to generate an animated display. Hence the process is called picture-driven animation.

The process is powerful because *it is easy to achieve rich variations in dynamic behavior by altering the data sequences while holding constant a few simple controlling algorithms.* The data sequences precisely determine the evolution of recurring picture change, within the constraints set by a choice of controlling algorithms.

We next introduce the three kinds of global dynamic descriptions, some useful algorithms for which they may be driving functions, and some useful methods for their static and dynamic pictorial representation and construction. The following classification will be helpful:

A *global dynamic description* is either
  a *movement description,* which is either
    a continuous movement description = a *path description,* or
    a discrete movement description = a *selection description;* or,
  a *rhythm description.*

Path descriptions

Consider those alterations of static pictures that consist of modifications of continuously variable parameters, such as location, size, and intensity. Their instantaneous values determine the picture's appearance at a given moment. Thus the static picture may be animated by specifying the temporal behavior of such parameters. A representation of the temporal behavior of a continuously variable parameter is called a *path description.*

The movement of a fixed-geometry picture (cel) in GENESYS is described as the change of two coordinates with time, and is represented by a pair of path descriptions. Their specification may be used to synthesize the drifting of a cloud, the zooming of a flying saucer, the bouncing of a ball, or the positioning of a pointer.

Since the behavioral descriptions of the parameters apply to entire intervals of time, the animation is liberated from a strictly frame-by-frame synthesis. *The computer is a medium through which one can bypass the static or temporally local and work directly on the dynamic or temporally global.* Movement is represented as it is perceived, as (potentially) continuous flow, rather than as a series of intermediate states.

Path descriptions, in fact, all dynamic descriptions, may be defined by one of six general approaches:

1. The sketching of a new pictorial representation of the description;
2. The editing or refinement of an existing pictorial representation of the description;
3. The direct algorithmic specification of the data sequence;
4. The indirect algorithmic specification in terms of existing data sequences;
5. An indirect algorithmic specification as a property of a constituent picture in an existing animation sequence; and,
6. A coupling to a real physical process in the external world, such that it transmits a data sequence as (analog) input to the computer. Interesting couplings may be to particle collisions, the atmospheric pressure, or, in the case of (1) and (2), a real live animator.

We shall in this paper be concerned with techniques implementing the first two approaches only. Sketching is useful when one knows the general shape and quality of a motion rather than an analytical expression for a function that determines it. Modifications of the sketches are frequently invoked after one views the current animation sequence and determines how it is inadequate.

There are two related kinds of pictorial representations of all movement descriptions, static and dynamic. Both kinds may be introduced with a single example.

Consider the motion of a figure that goes from one corner of a square room to the diagonally opposite corner by walking along two adjacent walls. We shall ignore the vertical movement and consider only motion of the center of the body in the two dimensions of the plane of the ground. He first walks in the direction of increasing X coordinate, then in the direction of increasing Y coordinate. We further assume that he begins from a standstill, accelerates and then decelerates to the first corner, pauses there for a brief interval while he turns in place, and finally accelerates and decelerates to his destination.

One complete description of this planar movement consists of the functions of the X and Y coordinates versus time. These are depicted in Figures 7 and 8. Such representations of changing picture parameters are called *waveforms*. Time is depicted, in the waveform, along one spatial dimension. The waveform's construction requires movement of the stylus along that dimension; the display records and makes tangible this movement.

Alternatively, both spatial coordinates could denote the two spatial coordinates of the movement. A natural correspondence is established between the X(Y) coordinate of the floor and X(Y) coordinate of the medium of the representation (paper, scope face, etc.). Figure 9 depicts such a *parametric curve* representation of the movement. It illustrates with clarity the figure's path on the floor.

Yet the dynamics of the motion are hidden because the temporal dimension is only an implicit coordinate. This rectified in Figure 10. A stream of symbols is used instead of a continuous trail to depict the path. Characters are spaced along the path at short, uniform intervals of time, such as every 24th of a second. Dynamics are apparent in the local density of symbols. Observe in particular how they cluster where the figure pauses.

The dynamic construction of a path description is a *user-driven animated display in which the timing of the stylus's movement is preserved by recording its position in every frame.* A tangible representation of the stylus path is the display of a sequence of characters spaced equally in time. We shall call a parametric curve dynamically sketched in real time a *p-curve.* The p-curve corresponding to Figures 7-10 is depicted in Figure 11. We have attempted to convey in a single static image that the p-curve is a dynamic display. Each 2-dimensional p-curve determines two path descriptions. Thus the hopping of the dog in 'DINNERTIME' may be synthesized by "hopping" with the stylus along some path on the tablet surface, that is by mimicking the desired dynamic.

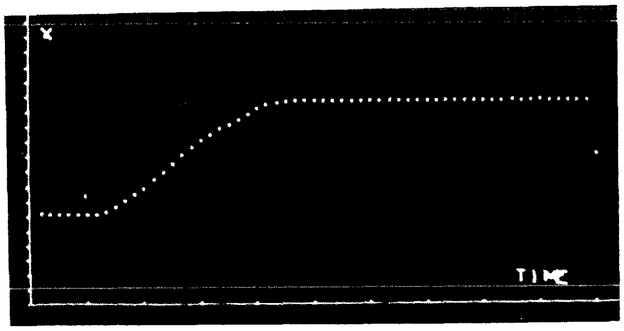


Figure 7—The X coordinate waveform of a movement

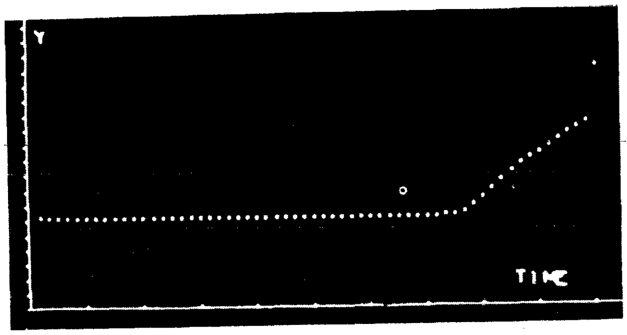


Figure 8—The Y coordinate waveform of a movement

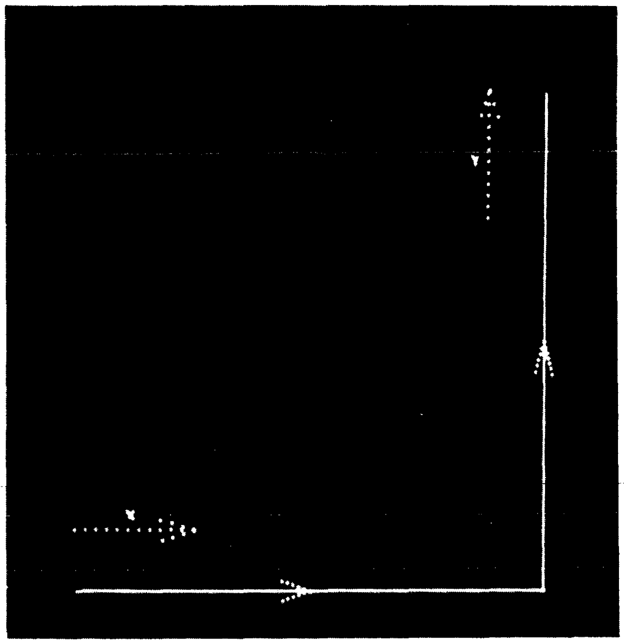


Figure 9—A parametric curve representation of the same movement. The rhythm of the movement is not visible

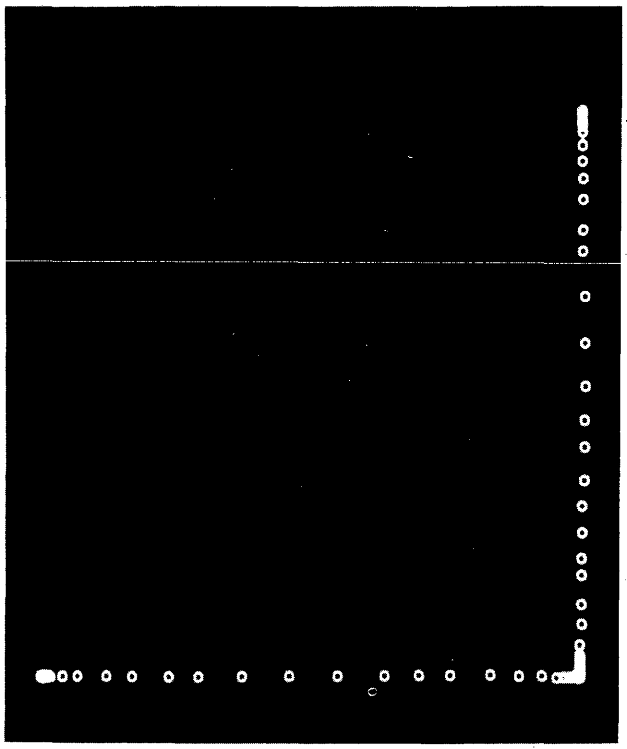


Figure 10—A better display of the parametric curve. Symbols are deposited at short, uniform intervals of time

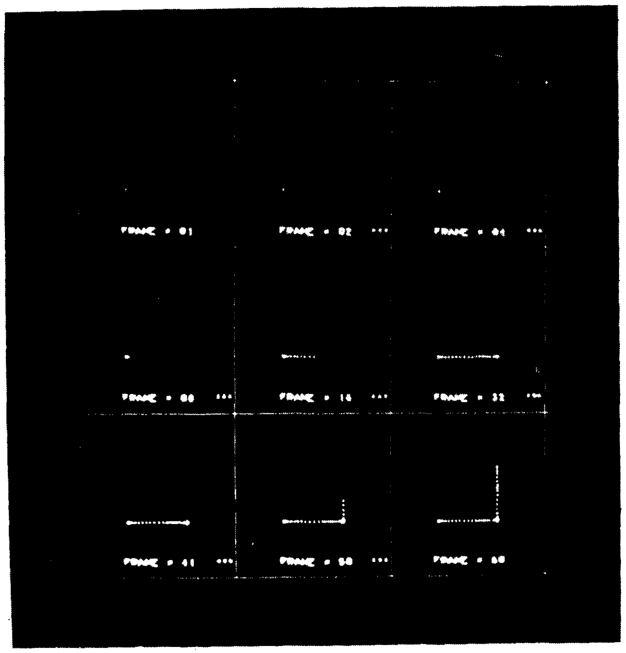


Figure 11—The p-curve corresponding to Figures 7-10. The dynamic display is compressed into a single static picture containing nine selected frames

In some cases one may need only one of the path descriptions. To depict the fluttering of a heart, we may assign the X coordinate of the p-curve to a parameter determining the size of the heart, and then flutter the pen back and forth horizontally. Any vertical motion that results is uninteresting and can be ignored.

A path description, in summary, defines dynamic activity that consists of potentially continuous and arbitrarily fine alterations of value. The reader should not be misled by the choice of the word "path". What is meant is a path, or sequence of values, through an arbitrary "continuous space", through a mathematical continuum. One application or interpretation of this path is the representation of a movement through the location-space of an object, such as a figure's path through a room. This interpretation, however, is not the only possible one. Depending upon the picture description capability of the system in which it is used, and the algorithm which it drives, *a path description may determine changing locations, intensities, thicknesses, densities, or texture gradients.* For example, a pulsating heart could be animated by varying either the size or the intensity of a single heart shape.

Reference 28 presents a detailed discussion of the relative strengths and weaknesses of waveforms, p-curves, and other static and dynamic representations of continuous movement. The discussion focuses on their uses as inputs of dynamics and as visual feedback to the animator, their dimensionality, their role in guiding temporal and spatial adjustments to existing motions, their capacity for conceptual extensions, and some practical problems (and solutions) that arise in the sketching process. Furthermore, we describe four kinds of editing and refining capabilities, operations for:

1. scaling curves;
2. shaping and reshaping them;
3. algebraically and logically combining them; and,
4. performing pattern scanning, matching, and transforming functions upon them.

Selection descriptions

Consider the algorithm that selects an element of the current frame from among members of a cel class. A good example arises in the synthesis of different facial expressions through the abstraction of discrete shapes and positions of mouth, nose, eyeballs, and eyebrows. One cel class could consist of the two members "eyebrows raised" and "eyebrows lowered." An animation sequence may be achieved by a temporal concatenation of selections from a cel class. A changing facial expression may be achieved by the parallel application of several such sequences of selections, one corresponding to each facial component. In 'DINNERTIME,' this technique was used to synthesize the movement of the dog's legs, tail, head, and tongue.

A representation of the dynamic selection from a finite set of alternative pictures is an example of the second type of global dynamic description and is called a *selection description.* The synthesis of selection descriptions is also aided by the use of pictorial representations, such as one consisting of a sequence of steps, where the length of each step is an integer multiple of frames, and the height is limited to transitions to and from positions on a discrete scale. Such pictures appear at the top of Figures 15 and 20. Superposition on a common time axis of pictures of several descriptions facilitates coordinating the counterpoint of the parallel selection strands.

The use of the term "selection" implies that a mechanism chooses from among a designated set of alternatives. In the previous examples the alternatives are cels, images to be introduced as components of frames in a dynamic sequence. A more general view of a selection description regards it as a sequence of *selectors,* functions which choose from a designated and finite yet potentially denumerable set of alternatives. Depending upon the picture description capability of the system in which it is used, and the algorithm which it drives, *a selection description may choose among alternatives that are subpictures, data, picture-generating algorithms, other global dynamic descriptions, pictorial events or activities, or strands of dynamic activity.* For example, the dynamic selection from among alternative picture-generating algorithms would be useful in a system with discrete texture choices, where there is one algorithm capable of filling an arbitrary region with that texture.

Further details may be found in reference 28, which also discusses techniques for the definition and editing of selection descriptions. These are conceptually similar to those used in the synthesis of path descriptions.

Rhythm descriptions

Rhythm descriptions consist of sequences of instants of display time (frames), or intervals between frames. They define patterns of triggering or pacing recurring events or extended picture change. In this context it is suggestive to think of a rhythm description as a pulse train. Each pulse may trigger the same action, or, as is discussed in reference 28, it may trigger one of several activities under the control of a selection description.

Rhythm descriptions facilitate the achievement of coordination and synchrony among parallel strands of dynamic activity. In this context it is suggestive to think of a rhythm description as a sequence of event markers. The marking sequence may be defined with respect to one pictorial subsequence, and then used to guide the construction of another subsequence.

A rhythm description cannot by itself define picture change; it can define a beat, a sequence of cues with respect to which picture change is temporally organized and reorganized. Animators have sometimes used metronomes as generators of rhythm descriptions.[2] Proper synchronization of a sound track to the visual part of a film is most critical to its success.[2]

Hence, rhythm descriptions marking critical instants of time play a key role in the synthesis and editing of movement descriptions. For these operations a rhythm description requires pictorial representation. In Figure 20 it is depicted both as a static pulse train and as a sequence of event markers along the axis of movie time. A direct and simple dynamic input, as we have seen in 'DINNERTIME', consists of tapping out the rhythm on a push-button.

Dynamic hierarchies

It is easy to conceive of more complex and useful couplings of global dynamic descriptions. Suppose, for example, that a hop, a skip, and a jump have each been synthesized with the aid of several path and selection descriptions. If the animator wishes to experiment with varying dynamic patterns of hop, skip, and jump, he should be able to define a selection description which chooses among these three alternatives. This is equivalent to defining selections among sets of path and selection descriptions. Reference 28 discusses the use of selection descriptions to establish arbitrary hierarchies of structured dynamic behavior, and illustrates the significance of this capability to the animator.

Exploratory studies in interactive computer-mediated animation

Three special-purpose picture-driven animation systems have been implemented on the M.I.T. Lincoln Laboratory TX-2 computer. A common feature is that each has a construction or editing mode, a playback or viewing mode, and a filming mode. In the first mode the animator may begin work on new pictures and global dynamic descriptions, or may recall and continue the construction of pictures and descriptions saved from other sessions. Algorithms embedded in the systems then compute TX-2 display files, in which sequences of frames composed of points, lines, and conic sections are encoded for use by the scopes.

These image files are passed to the playback program, which simulates a variable-speed, bi-directional, video tape recorder. The program normally sequences through the display file representation of successive frames, making each in turn visible for 1/24th of a second. One useful option is that of automatic cycling or the simulation of a tape loop.

When the animator has prepared a satisfactory sequence, he need no longer view it directly on the scope, but may instead want to record it on film. A pin-registered movie camera can be mounted in a light-tight box to a TX-2 scope. Its shutter is always open. The filming program (a variant of the playback program) "paints" an image on the scope. After a sufficient time interval to allow the decay of the phosphor, approximately 1/5 of a second, a signal from the computer advances the camera. A return signal upon the completion of the advance triggers the display of the next frame. The camera can be operated on one scope while we work at a tablet with another scope. Excellent film quality, with high contrast and low jitter, can be produced with the system.

The first two systems are very special-purpose. ADAM allows one to animate a crude line-drawing representation of a single human figure. EVE is an exercise in abstract dynamic art, in which one can animate a set of points linked by "rubber-band" straight lines. The animation technique in both cases is the specification, via waveforms and p-curves, of the seventeen path descriptions that define the temporal behavior of the picture's seventeenc ontrolling continuous parameters. A lengthy discussion may be found in reference 28; we shall here content ourselves with three observations:

1. *Clocked hand-drawn dynamics, or the dynamic mimicking of animated behavior, produces life-like, energetic movements,* even if used in ADAM to yield stick figure motions that are obviously not physically realizable, and even if used in EVE to yield abstract motions.
2. *Slight modifications to a waveform result in significant alterations to the character of an extended interval of a movement.* For example, ADAM's normal walk can be made into a jaunty saunter by the addition of more bounce to the vertical coordinate path description, or can be made effeminate by increasing the scale of the oscillations of the hip's rotational coordinate path description.

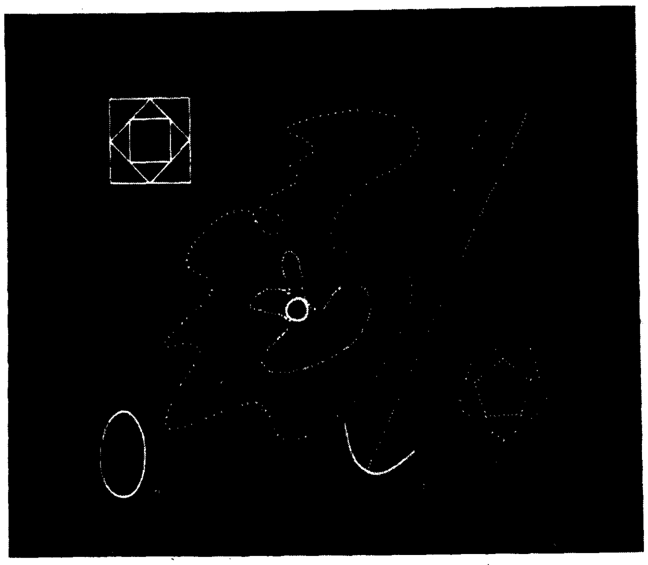


Figure 12—This picture, "drawn" by the author, illustrates the variety of line and texture that may be included in a GENESYS cel as of December, 1968 Free-hand sketches are portrayed by points spaced at an arbitrary, user-controlled density. Straight lines can be solid or can be dotted, over the same range of densities. Sections of circles, ellipses, parabolas, and regular polygons may be included. Arbitrary sub-pictures may be copied, translated, rotated, and scaled along two independent dimensions

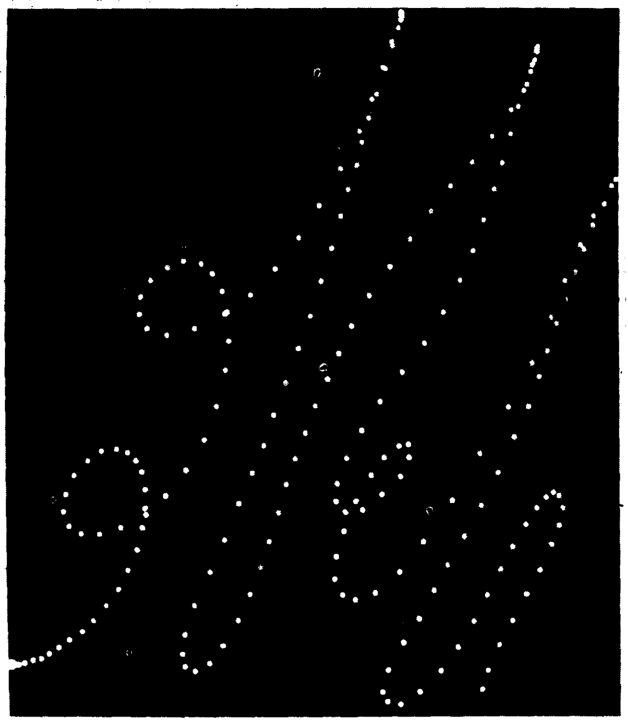


Figure 13—A parametric curve, the final frame of a p-curve, defining a movement that is life-like and energetic, smooth and graceful. Observe how points cluster at pauses in the motion

3. Even in a system whose only intended application is cartooning, *a dynamic mimicking capability must be augmented by an editing capability, for many motions cannot be mimicked or only so with difficulty, being purposeful exaggerations of real movements.*

Although GENESYS is also a special-purpose animation system, it is versatile enough to be used in the generation of a broad class of dynamic images. The term "generalized-cel," defined in reference 28, is a generalization of the concept of cel class illustrated in that its appearance in a given frame of the final dynamic display is determined by the values of a set of associated movement descriptions, both continuous and discrete.

The GENESYS animator may sketch, erase, copy, translate, rotate, and scale individual cels consisting of points, straight lines, and conic sections. He may sketch p-curves and dynamically tap rhythm descriptions. There are numerous tools for the manipulation of static representations of dynamic descriptions. Several individuals with varying degrees of artistic skill and training in animation have constructed short cartoon sequences with the aid of GENESYS. Figures 12–20 illustrate some of these experiences.

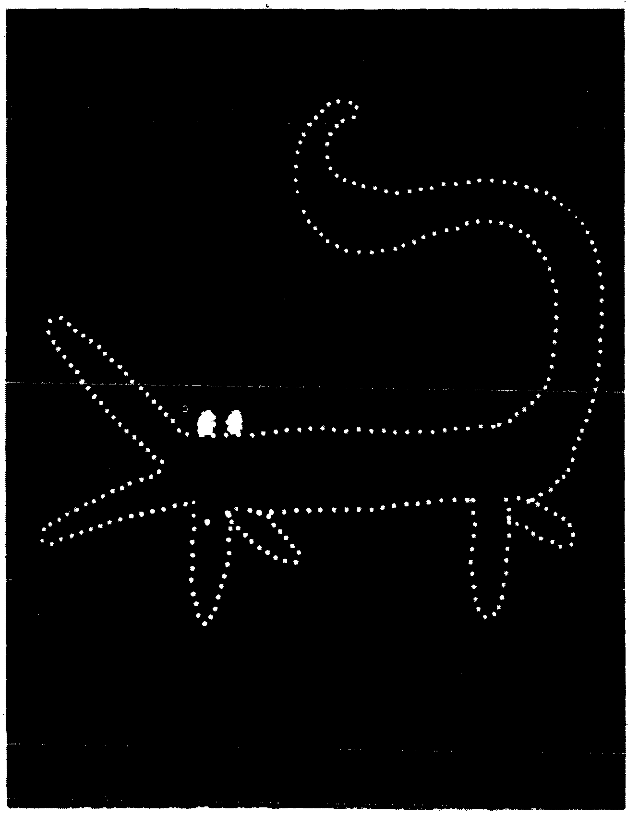


Figure 14—The crocodiless cavorts across the screen, delighted at her recent creation on the TX-2 console. The artist, Miss Barbara Koppel of Chicago, had little animation experience, no computer experience, a brief introduction to GENESYS, and assistance in using it from the author

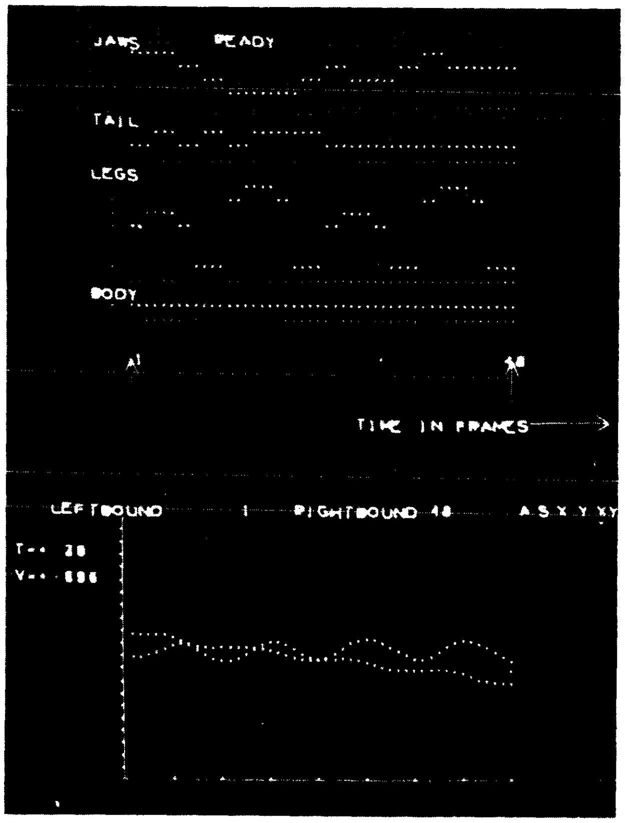


Figure 15—The four selection descriptions generate the movements of the jaws, tail, legs, and body of the crocodiless. Her translational motion is defined by the two path descriptions below. The oscillatory waveform is the vertical coordinate; the waveform sloping downward, the horizontal coordinate

Conclusion—the representation of dynamic information—The concept of a picture

Thus the essence of picture-driven animation is:

1. that there exists a set of abstractions of dynamic information, data sequences which drive algorithms to produce animated displays; and,
2. that these abstractions may in turn be modeled, generated, and modified by static as well as animated pictures, modeled in the sense that the picture structure represents the data sequence, generated and modified in the sense

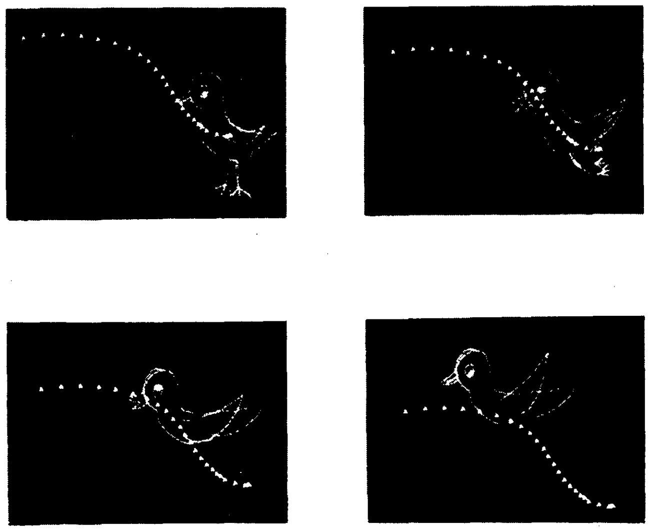


Figure 16—The 1st, 7th, 13th, and 19th frames of the 'take-off' of a bird are shown. The figure is superimposed on the parametric curve which defines its path through space. Mrs. Johnson has mimicked the motion by sketching the p-curve; the bird then reproduces this movement. Observe the switching among discrete shapes and positions of its eye, wing, and feet

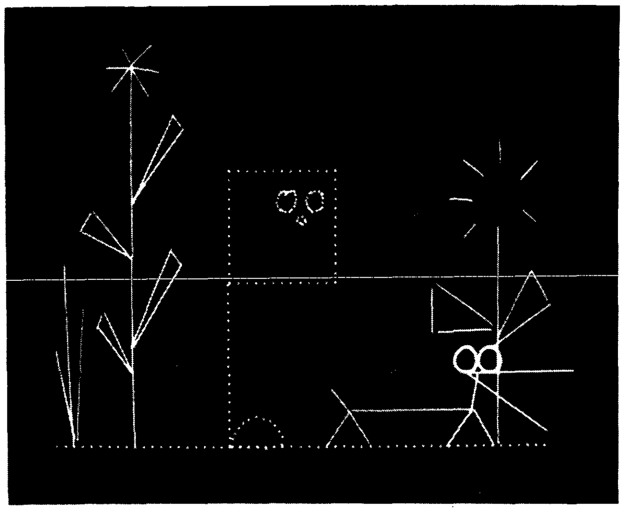


Figure 18—A short cartoon—what the viewer sees: A man, tripping blithely along, kicks a dog lying in his path. The dog rises and trots off to the right (shown above). It then returns, teeth bared (shown in Figure 19), and bites the man. The man jumps and runs away. The dog first follows, then returns once again to rest. The duration of the sequence is approximately 20 seconds

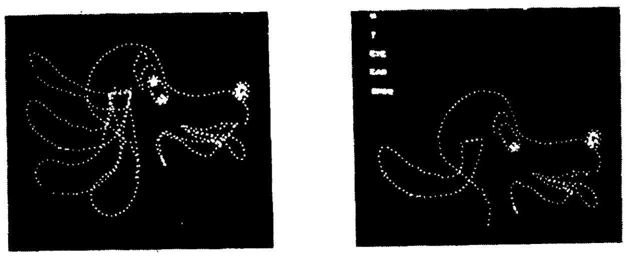


Figure 17—All cels used by Mrs. Johnson in the animation of Oopy—he flaps his ear, winks, and sticks out his tongue—are shown superimposed on the left. To the right GENESYS is in frame mode, in which the current state of a particular frame is displayed. Also visible are "light-buttons" representing cel classes (mouth, tongue, eye, ear, brow). The animator may alter the current frame, switching the selection of a cel from a class by pointing at it, or changing its position by turning knobs located under the scope. The underlying movement descriptions are automatically updated by GENESYS

that the picture represents the process of synthesis as well.

The three kinds of descriptions constitute *a rich, expressive, intuitively meaningful vocabulary for dynamics.* Each type abstracts an important category of dynamic behavior—flow and continuous change (path descriptions), switching and repetitive choice (selection descriptions), and rhythm and synchrony (rhythm descriptions). The vocabulary is economical, flexible, and general in the sense that it can characterize the dynamic similarities that exist in seemingly diverse animation sequences.

The use of dynamic descriptions couples picture definition by sketching and by algorithm; it furthermore allows both local (of the individual frame) and global (for an interval of time) control over dynamics. We have chosen to stress the latter and adopted the term "global dynamic description", for it is the capacity for global control that results uniquely from the use of the computer as an animation medium. Yet a dynamic description is not only a representation over an interval, but a sequence of single elements whose modification also provides local control over individual frames. Both local and global control are vital to the successful synthesis of movement. He who accidentally crashes into a wall while running from the police is going from the continuous to the discrete, from a global motion to a local event. He who aims to scale the wall is interpolating the continuous between the discrete, adjusting the global to fit the constraints of the local.

The naturalness and power of the vocabulary is increased by the ability to manipulate it in an interactive graphics environment. There exist, for each kind of data sequence, static pictorial representations

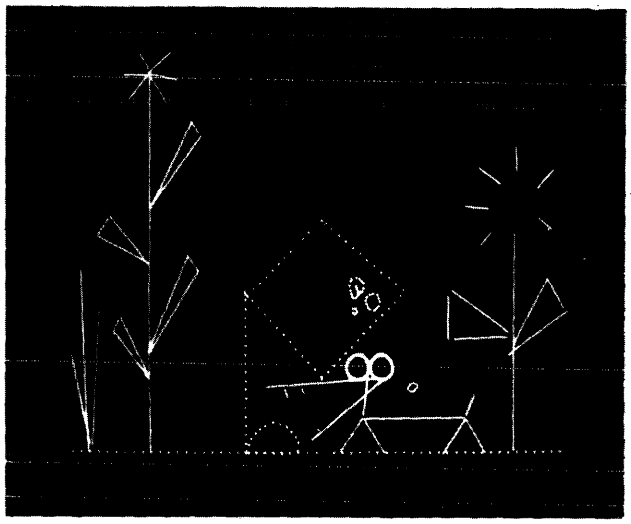


Figure 19—A short cartoon—how it was made: Mr Ephraim Cohen of Orange, New Jersey, a mathematician and programmer who is also a skilled caricaturist, completed the cels for his cartoon one week-end afternoon at the TX-2. The system then crashed, and he was forced to return home. He sent through the mail four selection descriptions, to choose cels from the classes "man's head", "man's legs", "dog's head", and "dog's body", and two path descriptions, to drive horizontally the man and the dog. The author input the dynamic descriptions, viewed the result, and then refined the movie by several iterations of editing the descriptions and viewing the sequence

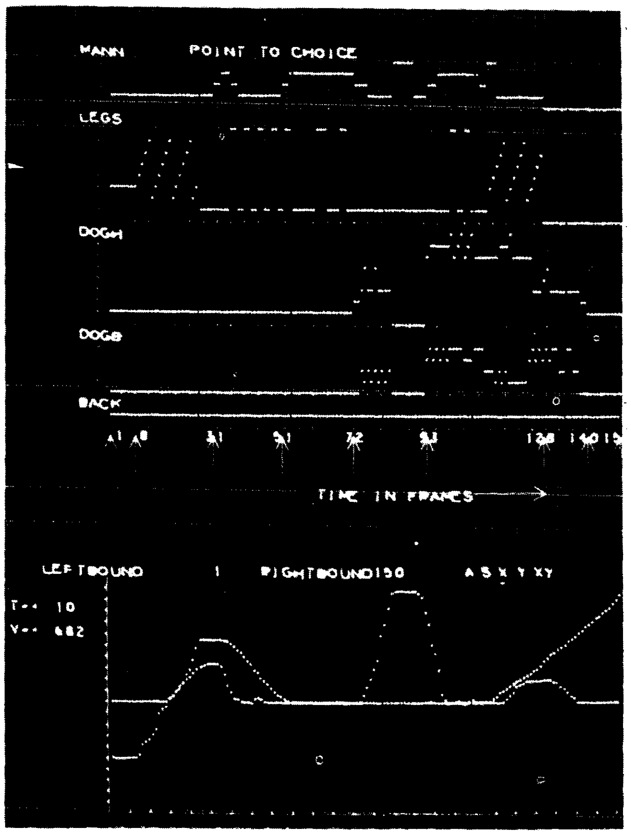


Figure 20—A short cartoon—why it works: The dynamic descriptions defining Mr. Cohen's cartoon as of January, 1969, are shown above. The selection descriptions, from top to bottom, belong to the man's head, the man's legs, the dog's head, and the dog's body. There are 4, 8, 8, and 4 cels in each class, respectively. The two waveforms represent the changes with time of the horizontal coordinates of the man and the dog

such as the waveform which provide a global view of and facilitate precision control of the temporal behavior implied by the sequences. There exist, for each kind of data sequence, methods of dynamic specification such as the clocked sketching of parametric curves which allow the animator's sense of time to be transmitted directly through the medium of the computer into the animated display.

We use the term "global dynamic description" and the names of the three types somewhat loosely in referring both to the underlying dynamic data sequences and to their corresponding pictorial representations. The imprecision is purposeful, for it is very significant that, in an interactive graphics environment, one can easily traverse in either direction any leg of the triangle {Dynamic Data Sequence, Static Pictorial Representation, Dynamic Pictorial Representation}. What results is an important plasticity in the representation of dynamics. Characterizations of change can be manipulated (shifted, stretched, superimposed, . . .) within and between the domains of the static and the dynamic. Several animation sequences can readily be related, coordinated, or unified, regardless of whether or not they ever occur concurrently. Dynamic behavior (data) can readily be transferred from one animation subsequence (including the animator) to another, from one mode of representation or embodiment in a picture to another.

Our concept of a picture is a broad one, and purposely so. For as we stress in reference 28, a computer-mediated picture is not only what is visible but what is contained in its model in the computer system. And the system, i.e., an interactive animation system, includes not only disks and core but an animator and perhaps an ongoing physics experiment as well as a tape-recorded speech. This system evolves continually through real time. Occasionally there occurs a particular reorganization of the system which results in the transfer of information from the animator to the pictorial data base, or in a computation on the data base which results in a sequence of visual images (i.e., data directly convertible by hardware into visual images). Thus, as we have stressed before, the act of mimicking dynamics is a (user-driven) dynamic picture. This *unification of the concepts of picture and action* is important.

The greater is the number and generality of available models of pictures and of processes of picture construction, the more flexible and powerful is the animation system in its ability to deal with dynamic information. The design of a multi-purpose, open-ended animation language that allows the animator himself to synthesize new models is outlined in reference 28. With such a language one can describe arbitrary action-picture interpreters that extract movement descriptions from the animator's use of system devices and transform them and existing static and dynamic displays into new static and dynamic displays.

Finally, the use of dynamic descriptions helps establish a conceptual framework which facilitates efficient use of the resources of the animation system: animator, software, and hardware. For details, we again refer the reader to reference 28.

Extensions, applications, implications

This paper is a pointer to a March, 1969, Ph.D. dissertation,[28] which includes the material contained herein considerably expanded, some suggestions for future research, and . . . .

1. There is a discussion of major difficulties in implementing systems embodying these ideas, with thoughts on the criteria supporting subsystems (both hardware and software) should satisfy to facilitate interactive computer-mediated animation. The environment in which current implementations exist is described in another paper being delivered at this conference.[32]
2. There is a lengthy outline of a proposed design of an Animation and Picture Processing Language. APPL is a multi-purpose, open-ended interactive animation programming language, through which the animator may also exercise algorithmic control over a dynamic display. The language will contain quasi-parallel flow of program control, a data structure that is a generalization of all hierarchic ordered data representations, an extensible class of picture descriptors, and a formalism which models the animator's dynamics as it models the dynamics of any picture, that is, as an integral component of animated system behavior. A major design goal is plasticity in the representation of dynamic information and flexibility in the techniques and conventions with which the animator interacts with the system. It has been verified on paper that a language containing these features can gracefully be used to construct dynamic displays, to build system tools that aid the construction process, and to implement special-purpose interactive computer-mediated animation systems.
3. Finally, there is a description of potential applications of this work in education, psychology, psychiatry, and the arts. In another paper being delivered at this conference, Huggins and Entwisle eloquently describe the role of computer animation in fulfilling the great untapped potential of "iconic modes of communication and instruction", in producing "visual images that in their ability to communicate ideas are superior to traditional graphical images on paper or blackboard."[33] "Instead of static images, words, and mathematical symbols", they suggest, "we may create dynamic signs that move about and develop in self-explanatory ways to express abstract relations and concepts." ". . . . A dynamic dimension is now available that requires the invention and development of new conventions and a visual syntax appropriate to this new medium if it is to be fully used for communication and education." May the ideas in our paper contribute towards this goal.

With respect to the arts, we conclude by repeating McLaren's description of animation:

"***Animation is not the art of *DRAWINGS-that-move* but the art of *MOVEMENTS-that-are-drawn.*

***What happens *between* each frame is more important than what exists *on* each frame.

***Animation is therefore the art of manipulating the invisible interstices that lie between frames. The interstices are the bones, flesh and blood of the movie, what is on each frame, merely the clothing."[34]

This paper may be regarded as a report on a use of the computer in "the art of *MOVEMENTS-that-are-drawn*," in the manipulation of "the invisible interstices that lie between frames."

ACKNOWLEDGMENTS

The encouragement, counsel, and insight of the dissertation's mentor, Professor Edward L. Glaser of Case Western Reserve University, and of Dr. William R. Sutherland of M.I.T. Lincoln Laboratory, Professor Murray Eden of M.I.T., and Mr. Eric Martin of Harvard University and Cambridge Design Group, Inc. are gratefully acknowledged. We appreciate the support of numerous individuals, here nameless but not forgotten, many in the Digital Computers Group of M.I.T. Lincoln Laboratory, who have contributed to the progress of this research.

REFERENCES

1 R STEPHENSON
Animation in the cinema
A Zwemmer Limited London A S Barnes and Co New York 1967

2 J HALAS R MANVELL
The technique of film animation
Hastings House New York 1959

3 K C KNOWLTON
A computer technique for producing animated movies
Proc S J C C 1964

4 K C KNOWLTON
A computer technique for the production of animated movies
Bell Telephone Laboratories Film

5 *The human use of computing machines*
Bell Telephone Laboratories Symposium June 20-21 1966

6 Conference on Computer Animation
Education Development Center Newton Mass July 17-18 1967

7 Proceedings of the 1967 UAIDE Annual Meeting

8 Proceedings of the Fall Joint Computer Conference 1968

9 F W SINDEN
Force, mass, and motion
Bell Telephone Laboratories Film

10 J L SCHWARTZ E F TAYLOR
Computer displays in the teaching of physics
Proc F J C C 1968

11 MIT SCIENCE TEACHING CENTER
Scattering in one dimension
Film available on loan from the Atomic Energy Commission

12 C LEVINTHAL
Molecular model-building by computer
Scientific American Vol 214 No 6 June 1966

13 C LEVINTHAL
Computer construction and display of molecular models
Film

14 E E ZAJAC
Computer-made perspective movies as a scientific and communication tool
Comm A C M Vol 7 No 3 March 1964

15 E E ZAJAC
Two-gyro, gravity gradient attitude control system
Bell Telephone Laboratories Film

16 S VANDERBEECK J H WHITNEY
Several animated films made with the aid of a computer

17 *Design and the computer*
Design Quarterly 66/67 Walker Art Center Minneapolis Minn

18 A M NOLL
The digital computer as a creative medium
IEEE Spectrum October 1967

19 J REICHARDT
Cybernetic serendipity, the computer and the arts
Studio International London and New York 1968

20 J C R LICKLIDER
Man-computer symbiosis
Trans IRE PGHFE HFE-1 4 1960

21 I E SUTHERLAND
Sketchpad: a man-machine graphical communication system
MIT Lincoln Laboratory Technical Report No 296 Jan 1963
Proc S J C C 1963

22 J C R LICKLIDER
Man-computer partnership
International Science and Technology May 1965

23 M A DAVIS T O ELLIS
The rand tablet: a man-machine communication device
Proc F J C C 1964

24 J F TEIXERA R P SALLEN
The sylvania data tablet
Proc S J C C 1968

25 L G ROBERTS
The lincoln wand
Proc F J C C 1966

26 J E CURRY
A tablet input facility for an interactive graphics system
Proc of the International Joint Conference on Artificial Intelligence 1969

27 E MARTIN
Private Communication

28 R M BAECKER
Interactive computer-mediated animation
Ph D Dissertation Department of Electrical Engineering MIT March 1969

29 T MIURA J IWATA J TSUDA
An application of hybrid curve generation—cartoon animation by electronic computers
Proc S J C C 1967

30 J NOLAN L YARBROUGH
An on-line computer drawing and animation system
Proc of the Conference of the International Federation for Information Processing (IFIPS) 1968

31 W H HUGGINS D R ENTWISLE
Exploratory studies of films for engineering education
Department of Electrical Engineering The Johns Hopkins University Report to US Office of Education September 1968

32 W R SUTHERLAND J W FORGIE M V MORELLO
Graphics in time-sharing: a summary of the TX-2 experience
Proc S J C C 1969

33 W H HUGGINS D R ENTWISLE
Computer animation for the academic community
Proc S J C C 1969

34 N MCLAREN
Quotation in animation exhibit in the Canadian Cinematique Pavilion at EXPO '68 in Montreal Canada
National Film Board Canada

Computer graphics displays of simulated automobile dynamics

by CALVIN M. THEISS

Cornell Aeronautical Laboratory, Inc.
Buffalo, New York

INTRODUCTION

Simulation of physical systems using digital computers has been accomplished by many people over the past few years. One such simulation program at Cornell Aeronautical Laboratory, Inc. constitutes an analytical representation of an automobile as it departs from the highway under various environmental conditions, especially under adverse ones where there is danger of collision with obstacles.[1] As with most complex simulation programs where there are interactions among many components, the equations of motion, including the required restraints, become long and numerous. But foremost, the output data set in printed form is quite extensive and very difficult for the investigating engineer to completely comprehend. About thirty crowded pages of output are printed for a five second automobile test run.

The objective of the task described here was to convert some of these output data into a pictorial form; that is, pictures were desired for a simulated automobile undergoing various violent maneuvers. In this manner, the actions in any part of the vehicle can be easily and quickly determined. Since this objective could be applied to most simulation problems to advantage, a set of generalized computer subroutines were designed to be called in a manner similar to most plotting utility packages.

The pictures are then drawn by state-of-the-art peripheral plotting hardware. Several output methods are available to CAL users—

a. on-line 8-1/2 × 11 inch pictures by Xerox equipment,
b. off-line CALCOMP plots on 30″ drawing paper,
c. off- or on-line film exposure through the use of a flying spot scanner built in-house, using a high precision cathode ray tube and a 16 mm movie camera. Commercial flying spot scanners are now available for this purpose.

Automobile simulation

The CAL Single Vehicle Accident Program[1] simulates a moving automobile, its interaction with the terrain over which it travels and collisions with a variety of obstacles in its path. The program is very flexible in that approximately 100 input parameters describe the automobile, such as wheel base, weight, braking coefficients, nonlinear characteristics of shock absorbers, engine torque on driver wheels, etc. Likewise, various terrain profiles and obstacles can be specified for different simulation runs. For example, the radial tire force generated at each wheel is simulated by means of simple springs on smooth terrain; however, on rough terrain or when obstacles, such as curbs, are encountered, the tires are simulated by a set of compound springs radiating from the hub of the wheel. Originally, the output consisted of about sixty items of response printed in tabular form.

For a first effort at pictorial output, the automobile is considered as four different objects, namely, the two front wheels, the rear axle and the sprung mass (the body). The Single Vehicle Accident Program was modified to record dynamically on magnetic tape thirteen parameters which describe the actions of these four objects. They are—

1-3	position of the sprung mass,
4-6	attitude of the sprung mass,
7-8	deflection of each of the front wheels,
9-10	camber of each of the front wheels,
11	deflection of the rear axle,
12	roll of the rear axle,
13	steer angle applied to the front wheels.

This magnetic tape is the dynamic input to the picture generation program. Other objects, such as road markings, roadside barriers, ramps, etc., are static initial inputs to the program.

Perspective picture generation

A quick but incomplete review of the available literature revealed that a number of different approaches to producing three-dimensional computer graphics displays have been developed.[2,3,4,5] However, none met our particular requirements to permit quick economical generation of both still and motion pictures with the existing in-house equipment available.

The pictures are drawn as line drawings using straight lines only. The periphery of a wheel, for example, is drawn as a 50-sided regular polygon. No attempt was initially made for "hidden line removal".

Description of objects

Each object is stored as one or more three-dimensional arrays; each entry defines a point of the object with respect to three-dimensional rectangular coordinates fixed in the object. Thus, after the appropriate coordinate transformations discussed below, a perspective picture can be drawn in two-dimensional space by starting at the first point of the first array, drawing a straight line to the second point and proceeding onto each point of the array in turn. This is done for each array of the object.

Camera simulation

The usual description of a simple pinhole camera as illustrated in Figure 1 is used. The position of the image, Q, on the picture plane of an object at point P in front of the camera with a focal length of F is:

$$x_Q = 0 \quad (1)$$

$$y_Q = -F\, y_p/(x_p - F) \quad (2)$$

$$z_Q = -F\, z_p/(x_p - F) \quad (3)$$

Thus, the three-dimensional space in front of the camera is mapped onto the two-dimensional space of the picture plane (film) of the camera. The simulation of this simple camera is the crux of our picture generating program.

In addition, care must be taken to assure that attempts are not made to draw portions of the object image which are outside the picture boundary. At first thought one might expect this to occur only when

$$|y_Q| \geq Y_{max} \quad (4)$$

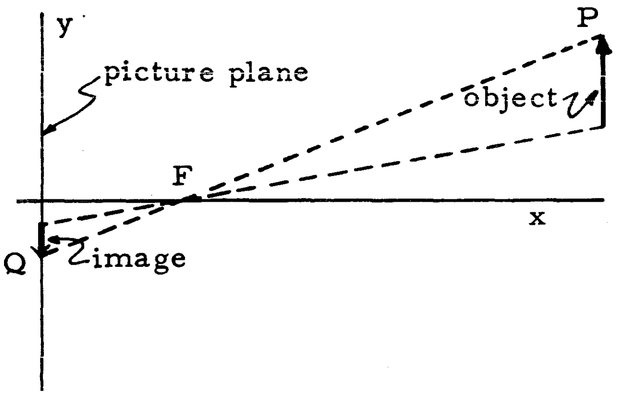


Figure 1—Diagram of a pinhole camera

or

$$|z_Q| \geq Z_{max} \quad (5)$$

Actually this occurs also as

$$x_p - F \to 0 \quad (6)$$

For drawing a single straight line, only the central portion, an end, or the whole line may be in view. Thus, intersections of lines and picture edges must be continually tested. For example, see Figure 2.

Line AB is defined in the object array by points A and B, the transformations of both lie outside the picture. However, the central portion of the line lies inside the boundary and must be drawn. Thus, for lines where at least one of the end points lies outside the boundary, algorithms were developed to determine which portion, if any, of the line is to be drawn.

Definitions of coordinates and parameters

As noted above, coordinate transformation of the arrays of points of an object in three-dimensional space

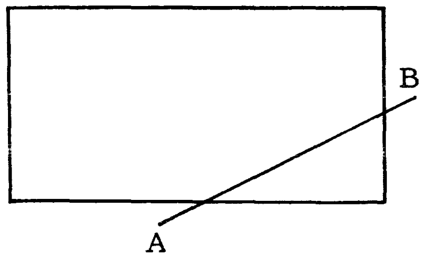
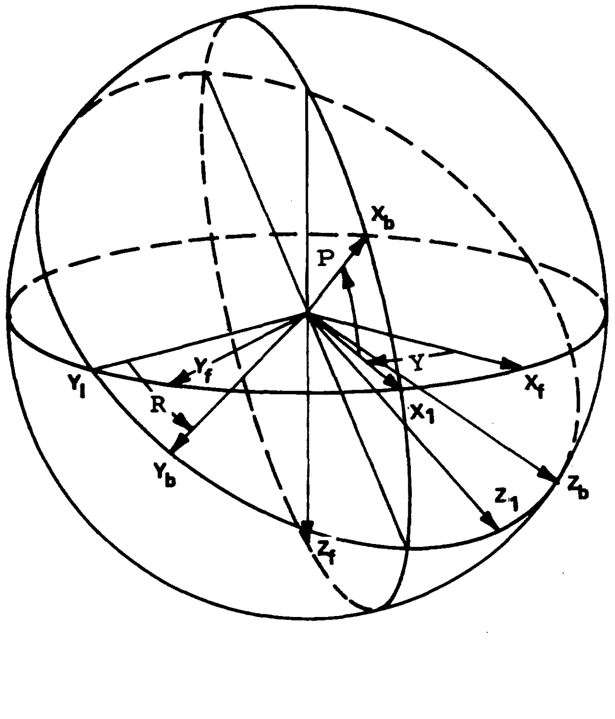


Figure 2—Image line intersecting with picture boundary

to points of the picture in two-dimensional space is necessary. This is done with the help of the following coordinate systems and relating parameters.

a. *Inertial coordinate system*: a rectangular coordinate system fixed in free space. This is the fundamental reference base for spatial relationships among the various components.
b. *Object coordinate system*: a rectangular coordinate system fixed to the object. One exists for each object.
c. *Camera coordinate system*: a rectangular coordinate system fixed to the camera, with the origin in the center of the picture plane and the positive X axis extending toward and through the focal point.
d. *Picture coordinate system*: a two-dimensional rectangular coordinate system fixed to the picture frame.
e. *Object position*: the position of the origin of object coordinates, with respect to the inertial coordinate system (the row vector, **B**).
f. *Object attitude*: the angular relationship of the object coordinates with respect to the inertial coordinates in terms of Euler angles, yaw, Y; pitch, P; and roll, R (see Figure 3).
g. *Camera position*: the position of the origin of the camera coordinates with respect to the inertial coordinate system (the row vector, **C**).
h. *Camera attitude*: three angles, the azimuth, A, and elevation, E, of the camera line of sight (camera X axis) and the orientation of the picture (tilt), T, with respect to the inertial coordinates.
i. *Camera focal length*: (defined in Figure 1).



Y	about axis	Z_f	
P	about axis	Y_1	(An Intermediate Axis)
R	about axis	X_b	

Figure 3—Euler angles relating body axes (X_b, Y_b, Z_b) w th respect to fixed axes (X_f, Y_f, Z_f)

Spatial relationships

Motion and dynamic activity in the simulated scene to be "photographed" is reflected in changes over time of the position and attitude parameters (a total of six) for each object. Furthermore, change in the camera parameters are also possible. The camera could be mounted on a vehicle. Panning (attitude changing) and/or zooming (changes in focal length) are included.

Consider a point of an object, any point in the object description arrays, as a row vector, $\mathbf{P}_0$, described in object space. The position vector with respect to inertial space, is obtainable by the transformation.

$$\mathbf{P}_I = \mathbf{P}_0 \times [\mathrm{OI}] + \mathbf{B} \qquad (7)$$

where the transformation matrix is

$$[\mathrm{OI}] = \begin{bmatrix} \cos P\cos Y, & \sin P\cos Y\sin R - \cos R\sin Y, & \sin P\cos Y\cos R + \sin R\sin Y \\ \cos P\sin Y, & \sin P\sin Y\sin R + \cos R\cos Y, & \sin P\sin Y\cos R - \sin R\cos Y \\ -\sin P, & \cos P\sin R, & \cos P\cos R \end{bmatrix} \qquad (8)$$

Similarly, in camera space, the relationship is

$$\mathbf{P}_c = (\mathbf{P}_I - \mathbf{C}) \times [\mathrm{IC}] \qquad (9)$$

where the transformation matrix is

$$[\mathrm{IC}] = \begin{bmatrix} \cos A\cos E, & \sin A\cos E, & \sin E \\ -\sin A\ , & \cos A\ , & 0 \\ -\cos A\sin E, & -\sin A\sin E, & \cos E \end{bmatrix} \qquad (10)$$

At this point the value of the x component of $\mathbf{P}_c$, x_{pc}, is checked. If $x_{pc} \leq F$ (see Figure 1), the point is (1) behind the camera, or (2) at the camera focal point, or (3) its corresponding picture image point will be a very large distance from the center of the picture, i.e., x_Q,

$y_Q \rightarrow \infty$. Thus, complete lines connecting to the point cannot be drawn.

The final transformation to the two-dimensional point in the picture plane is

$$\mathbf{Q} = \frac{KF}{x_{pc} - F} \mathbf{P}_c \times [CP] \tag{11}$$

where K is an enlargement factor and the transformation matrix is

$$[CP] = \begin{bmatrix} 0 \;, & 0 \\ -\cos T, & \sin T \\ \sin T, & \cos T \end{bmatrix} \tag{12}$$

Now the point must be checked to determine whether it is within the picture frame by the following criteria—

$$|x_Q| \leq W/2 \tag{13}$$

$$|y_Q| \leq H/2 \tag{14}$$

where W and H are the prescribed width and height of the picture, respectively.

The picture generation program

The program was written using the FORTRAN IV source language and documented for general usage[6] in the analysis of automobile dynamics.

Picture subroutine set

The subroutines were designed to supply the rudimentary requirements for drawing perspective line drawings. A brief description of each subroutine call follows.

a. CALL FRAME
   1. When output is to the flying spot scanner, the film is advanced one frame.
   2. When output is on the CALCOMP plotter, a new area of the paper is selected.

b. CALL FRAMSZ (W, H)
   The size of the picture is established.

c. CALL CAMSET (POS, ITYPE, INVT)
   The camera parameters are established.

   POS(n) n=1,2,3 specifies the camera position, **C** .
   POS(n) n=4, 5, 6, specifies the position of the camera focal point with respect to the inertial coordinates if ITYPE=1.
   POS(n) n=4, 5, 6 specifies the azimuth and elevation of the camera line of sight and the camera focal length if ITYPE=2.
   POS(7) specifies the rotation of the picture about the line of sight, i.e., its tilt angle.
   INVT specifies if either the vertical or horizontal axes (or both or neither) is to be reversed.

d. CALL ØBAXIS (X, Y, Z, ROLL, PITCH, YAW)
   The position and attitude of the object axis are established. All further calls to OBLINE refers to this object coordinate system until another call to OBAXIS occurs.

e. CALL ØBLINE (X, Y, Z, N)
   X, Y, Z are arrays containing the respective components of N points with respect to the object coordinate system established by the last call to OBAXIS. All pertinent coordinate transformations are computed and a perspective line drawing is made connecting the N points in the order of the array. The drawing is "clipped" wherever it crosses the picture boundary as established by the previous call to FRAMSZ.

f. CALL CIRCLE (X, Y, Z, R, AZ, EL, TI, P, N)
   The data for the arrays required by a call to OBLINE are prepared for drawing a circle centered at point X, Y, Z with radius R. The circle axis points in the direction with an azimuth AZ and elevation EL. Actually, the circle is approximated by an N-sided regular polygon. P is the returned data arrays. TI specifies the rotation or starting point of the polygon; for a true circle it has no significance.

g. CALL ØBJECT (TITLE, X, Y, Z, ROLL, PITCH, YAW)
   This call and the next two calls are included only to reduce the bookkeeping details for the user. TITLE is the name of an object to be drawn, whose object axis is defined by the last six arguments. The descriptive arrays for this object are stored with the object name in a common store. When the ØBJECT is called, the store is searched for the object named, the coordinate transformations calculated and the object is drawn. Subroutines ØBAXIS and ØBLINE are called by this subroutine.

h. CALL ØBJINP
   This call reads data from cards giving the name and descriptive arrays of objects and stores them in the common store.

i. CALL DLTØBJ (TITLE)
   The object TITLE is deleted from the common store, thus freeing storage space for new objects.

The executive routine

The executive routine reads the magnetic tape describing the thirteen parameters of the automobile dynamics. These in turn are used for determining the spatial relationships of the object coordinates for each of the four objects (the two front wheels, rear axle and sprung mass) of the automobile for each increment in time. Calls to the picture subroutines are then made to draw the pictures. Background objects such as road markings, barriers, ramps, etc., are read in from cards. Other features of the program include:

a. The frame rate (or time increment between pictures) is specified for each usage of the program. In fact, it may be changed at any point in the run. In this manner several "still pictures" may be chosen at arbitrary times throughout the simulated event.
b. Background objects may be added or deleted at any point in time.
c. Camera motion, if desired, may be specified or changed at any point in time. Further, panning and/or zooming, if desired, may be specified with given rates or automated to keep the automobile in the center of the picture.
d. The program can also write titles and subtitles in the film where desired.

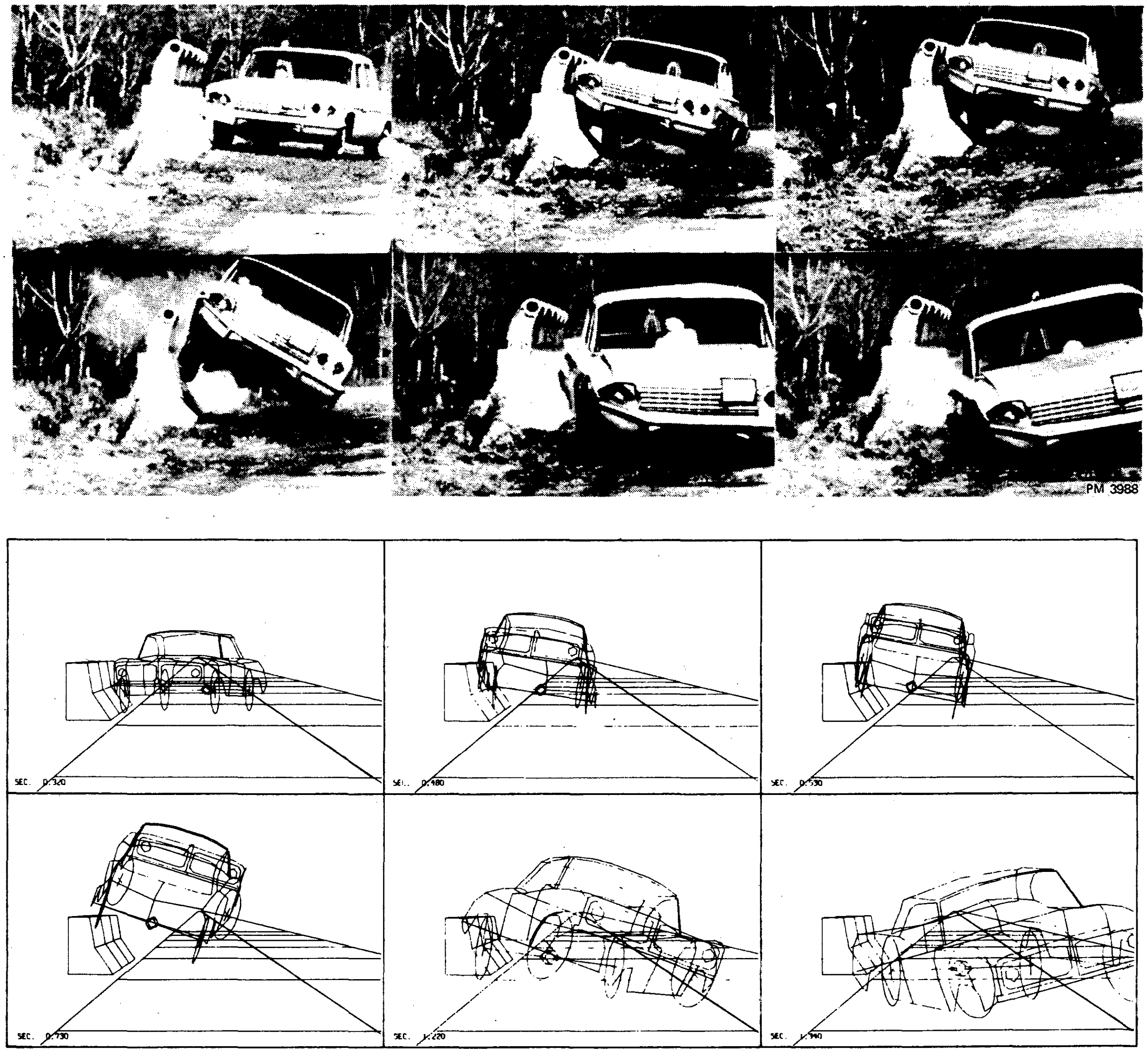


Figure 4—Photographic and computer graphic display of GM parapet impact

Sample applications

Sample still pictures from several different simulated vehicle maneuvers are shown in Figures 4, 5 and 6, together with pictures of similar "real life" tests. Automatic panning is used in all cases.

CONCLUSIONS

The pictures illustrate the ease of assessing the capability of the simulation program. Through comparisons with similar "real life" test pictures, especially motion pictures, one may quickly verify the validity of a simulation program and establish a high degree of confidence in its usage.

These results encourage one to look ahead to future activity. The various other applications of this output technique to simulation is limited only by the user's imagination and needs. Improvements, of course, are desirable. First of all, the removal of hidden lines would improve the esthetic value. The literature contains several methods of varying levels of satisfying and financially feasible results.[7,8,9] The economical inclusion of color and shading is challenging. Illustration of the deformation of objects, such as the crushing of an automobile fender, would be highly desirable.

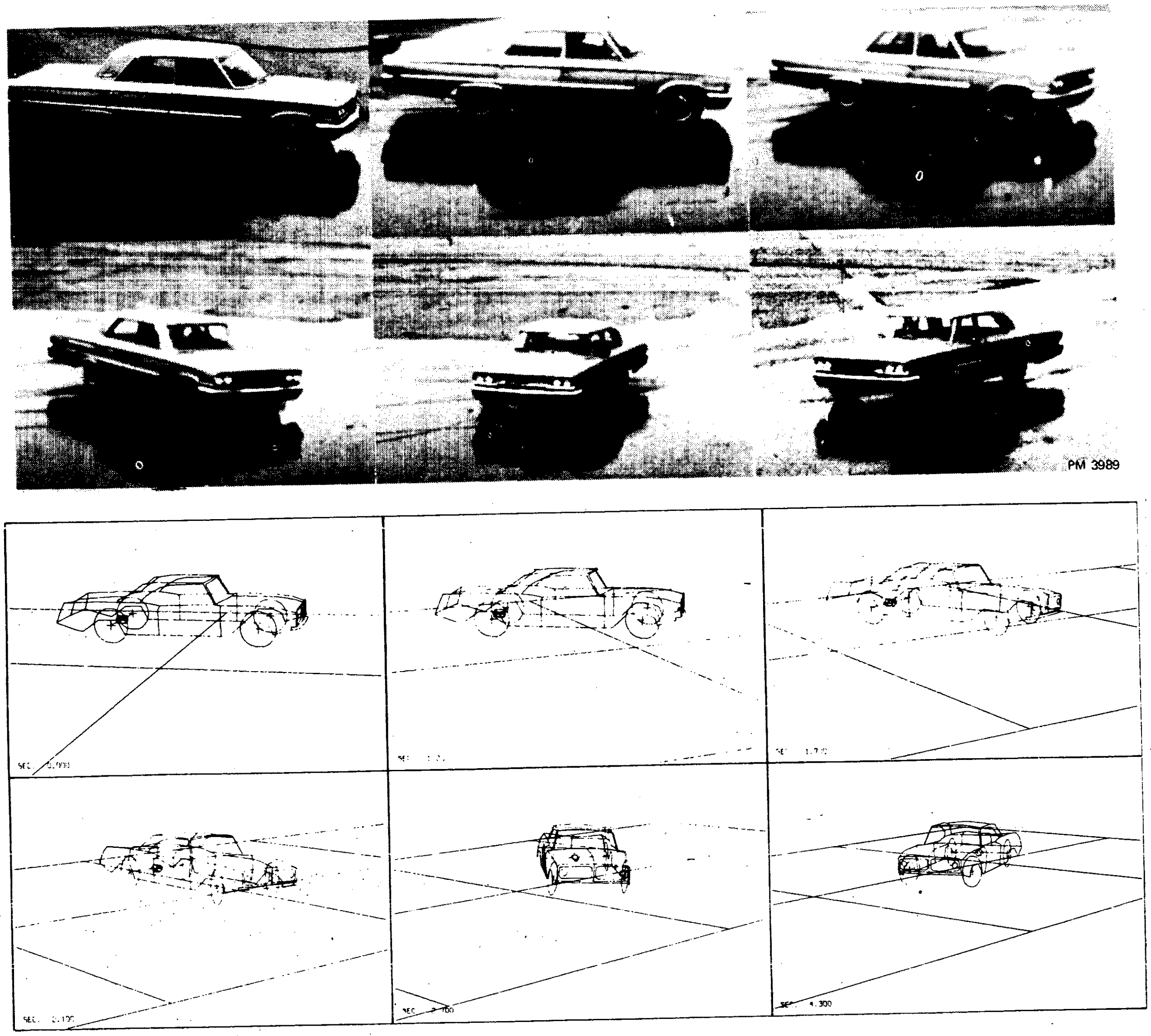


Figure 5—Photographic and computer graphic display of skidding vehicle

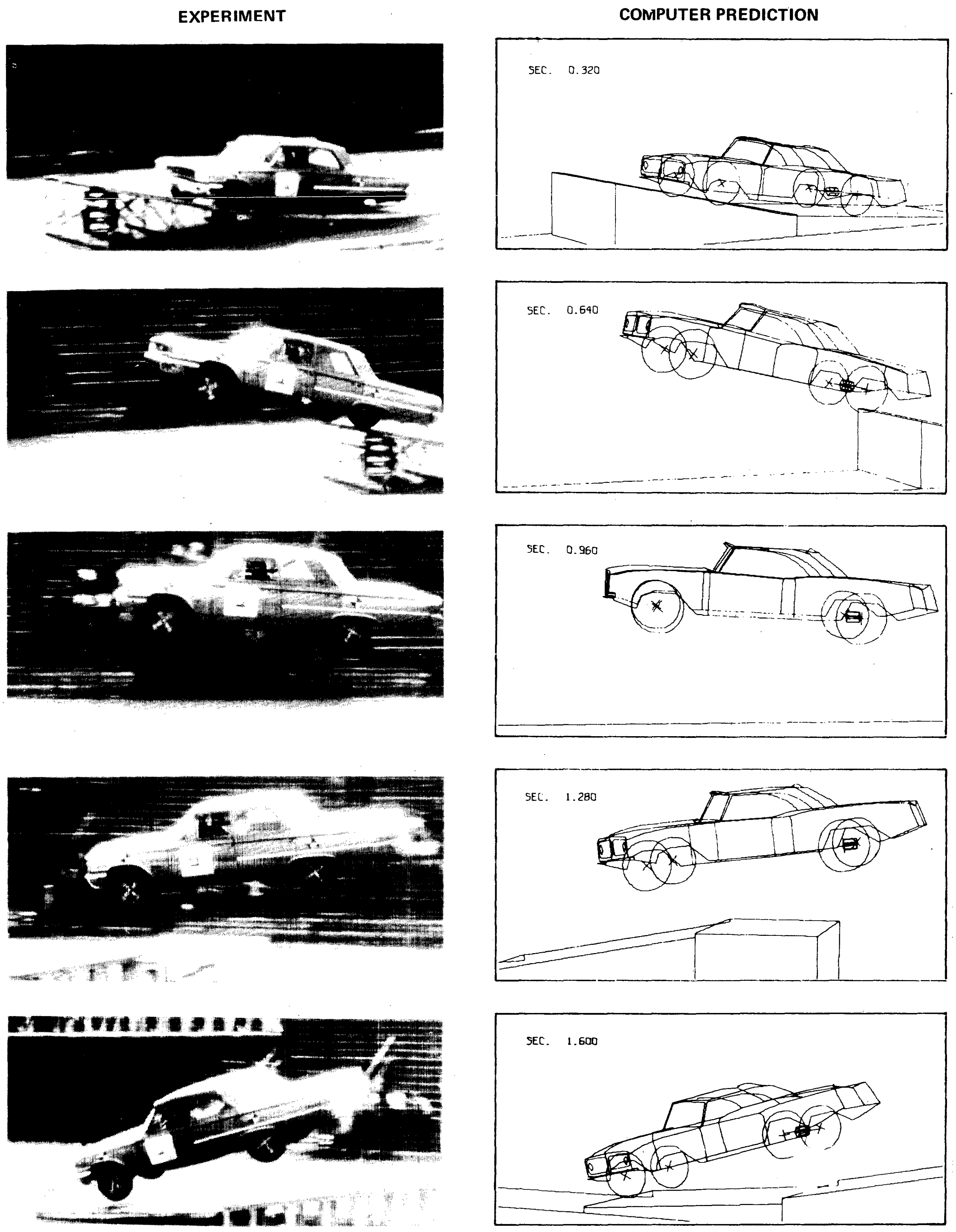


Figure 6—Ramp jump at 44 mph

ACKNOWLEDGMENTS

I wish to express my gratitude to Raymond R. McHenry and Claron W. Swonger for their help and guidance in the development work leading to this paper.

The reported research was performed under Contract No. CPR-11-3988 with the Traffic Systems Division, Office of Research and Development, Bureau of Public Roads, U. S. Department of Transportation. The opinions, findings and conclusions expressed in this paper are those of the author and not necessarily those of the Bureau of Public Roads.

REFERENCES

1 R R McHERNY N J DeLEYS
Vehicle dynamics in single vehicle accidents—validation and extensions of a computer simulation
Cornell Aeronautical Technical Report VJ-2251-V-3 December 1967

2 K G KNOWLTON
A computer technique for producing animated movies
Proc AFIPS Conference 1964

3 A M NOLL
Computer-generated three-dimensional movies
Computers and Automation November 1965

4 W A FETTER
Compu er graphics
Annual Meeting of the American Society for Engineering Educators 1966

5 R L MITCHELL
A computerized 3-D plotting program
Computerized Imaging Techniques, Proc of the Society of Photo-Optical Instrumentation Engineers June 1967

6 C M THEISS
Perspective picture output for automobile dynamics simulations
Cornell Aeronautical Laboratory Technical Report VJ-2251-V-2R January 1969

7 L G ROBERTS
Machine perception of three-dimensional solids
Lincoln Laboratory Technical Report 315 May 1963

8 R A WEISS
BE VISION, a package of IBM 7090 FORTRAN programs to draw orthographic views of combinations of plane and quadric surfaces
J ACM April 1966

9 P LOUTREL
A solution to the hidden-line problem for computer-drawn polyhedra
New York University School of Engineering Technical Report September 1967 400-167

Fast drawing of curves for computer display*

by DAN COHEN** and THEODORE M. P. LEE**

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Now that computer displays with vector and character drawing capabilities are becoming a common form of on-line graphic output device, interest has turned toward providing a curvilinear display capability. Naturally, curves can be drawn as a series of short vectors stored in display memory, but the goal of current research is to provide hardware which draws a curve by generating such vectors, or perhaps, continuous beam motion, from a concise specification of a curve segment.

In this paper we consider the properties of a particular class of curves suitable for such a hardware curve generator. (See Figures 1 and 2 for examples.) As it happens, the algorithms for generating successive points on the curve can be implemented efficiently in software. This work is derivative in part from suggestions by S. A. Coons and in part from ideas developed during the design of the Harvard Three-Dimensional display.[1,2] In both cases a direct debt to L. G. Roberts, then of Lincoln Laboratories, must also be acknowledged.

Mathematically we present this material in as general a form as seems appropriate for the particular topic under discussion. Practically we think primarily of two dimensional curves or three dimensional curves projected into two dimensions. In this and related work on the display of three-dimensional surfaces it has proven convenient to formulate the concepts in a homogeneous coordinate geometry.[3] This formulation is especially relevant when talking about perspective images, for it was in the discussion of projective geometry in the late 19th century[4,5] that such a representation for projective spaces was introduced. As is to be expected, most of the geometrical results derived at that time do not seem relevant to computer applications, for they normally involve deductive rather than constructive proofs.

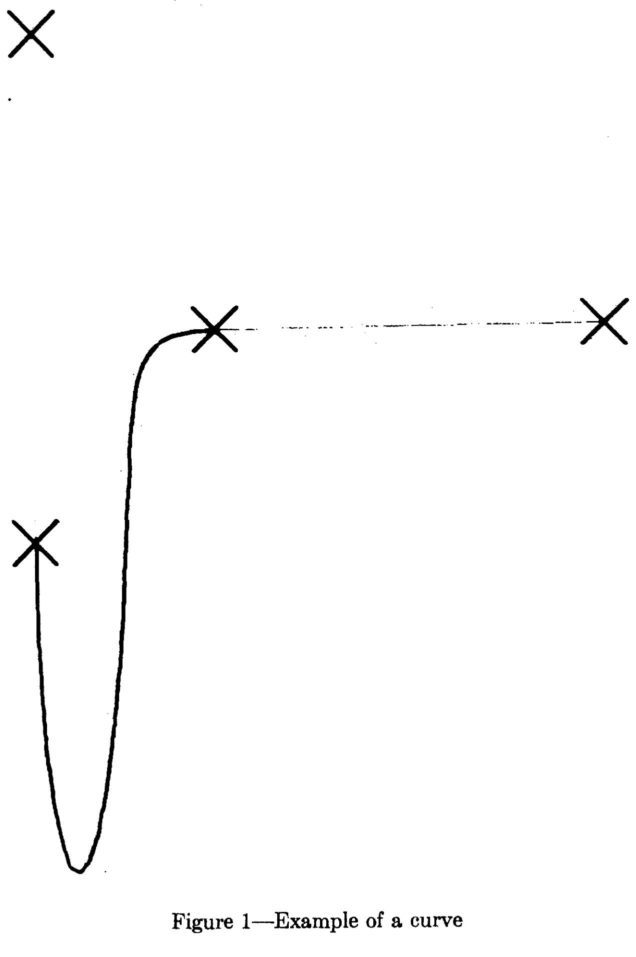


Figure 1—Example of a curve

* The research reported here was performed at Harvard University and supported in part by the Advanced Research Projects Agency (ARPA) of the Department of Defense under contract SD-265 with Harvard University, under contract AF 30(602)-4277 with the University of Utah, by the Office of Naval Research under contract ONR 1866(16) with Harvard University, and by a long standing agreement between Bell Telephone Laboratories and the Harvard Computation Laboratory. Mr. Lee is a National Science Foundation graduate fellow.

** At the Aiken Computation Laboratory, of the Division of Engineering and Applied Physics.

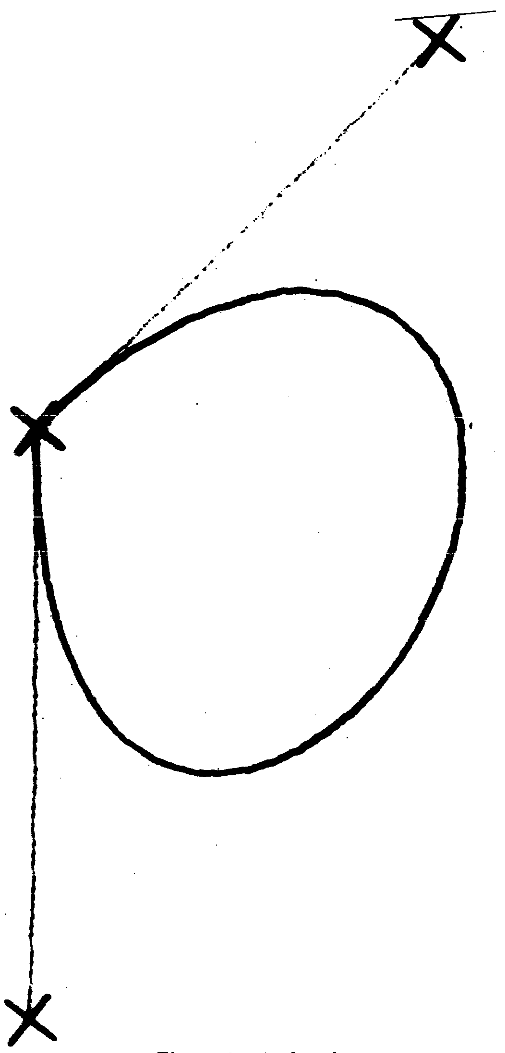


Figure 2—A closed curve

We consider the family of curve segments

$$\Phi = \{V = V(t), 0 \leq t \leq 1\}$$

where each component of **V** is a polynomial in t. We restrict the discussion to polynomials only because of the simplicity of polynomial evaluation. We consider these curves in R^n; however our interest lies in P^n, the perspective space of dimension $n - 1$, generated from R^n by dividing each component of V by the last component.

Let Φ_m be the family of curve segments defined by polynomials of degree not exceeding m. The higher m is, the more general is the family Φ_m, but generating each point on any curve is more complex. The greater generality of Φ_m for larger m enables the curve to satisfy more conditions, but correspondingly requires more conditions to uniquely define the curve. Φ_2 includes straight lines and conic sections only. Therefore if $V \epsilon \Phi_2$, V cannot cut itself and cannot have inflection points. Φ_3 (which includes Φ_2 as a proper subset) contains curve segments which may have inflection points and which may cut themselves, as shown in Figures 3 and 4.

For display purposes, on a two-dimensional scope face, it is enough always to work in P^3, although for many practical uses one wants to impose conditions on the curve to be satisfied in R^n. In particular, one often wishes to characterize a curve to be displayed in two-dimensional projection by conditions expressed in the projective space P^4, that is, in three-dimensional homogeneous coordinates.

Iterative generation of curve

There are several methods for generating a curve, depending upon its parametric representation. The simplest is to generate a sequence of short line segments between points on the curve corresponding to successive values $\{t_0, t_1, t_2, \ldots, t_N\}$ of the parameter t, with a fixed difference $\delta = 1/N$ between t_m and t_{m+1}. Let $x(t) = X(t)/W(t)$ and $y(t) = Y(t)/W(t)$. Let P(t) represent any of the polynomials X(t), Y(t), or W(t) and let $P_n = P(t_n) = P(n\delta)$ for $n = 0, 1, 2, \ldots, N$. In the usual fashion we define two difference operators with respect to the difference δ as follows:

the forward difference: $\Delta f(x) = f(x + \delta) - f(x)$

the backward difference: $\nabla f(x) = f(x) - f(x - \delta)$

Δ^m and ∇^n are defined recursively and $\Delta^0 f = \nabla^0 f = f$.

Note that $P_{n+1} - P_n = \Delta P_n = \nabla P_{n+1}$ and that if

$P(t) = \sum_{k=0}^{m} a_k t^k$ then $\Delta^m P = \nabla^m P = m!a_m$ for any value of t. For the convenience of the presentation we will let m = 3 in the following.

It is easy to see that

$$\begin{bmatrix} P_{n+1} \\ \Delta P_{n+1} \\ \Delta^2 P_{n+1} \\ \Delta^3 P_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_n \\ \Delta P_n \\ \Delta^2 P_n \\ \Delta^3 P_n \end{bmatrix}$$

or, in short, $\mathbf{F}_{n+1} = S\,\mathbf{F}_n$
and

$$\begin{bmatrix} P_{n+1} \\ \nabla P_{n+1} \\ \nabla^2 P_{n+1} \\ \nabla^3 P_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_n \\ \nabla P_n \\ \nabla^2 P_n \\ \nabla^3 P_n \end{bmatrix}$$

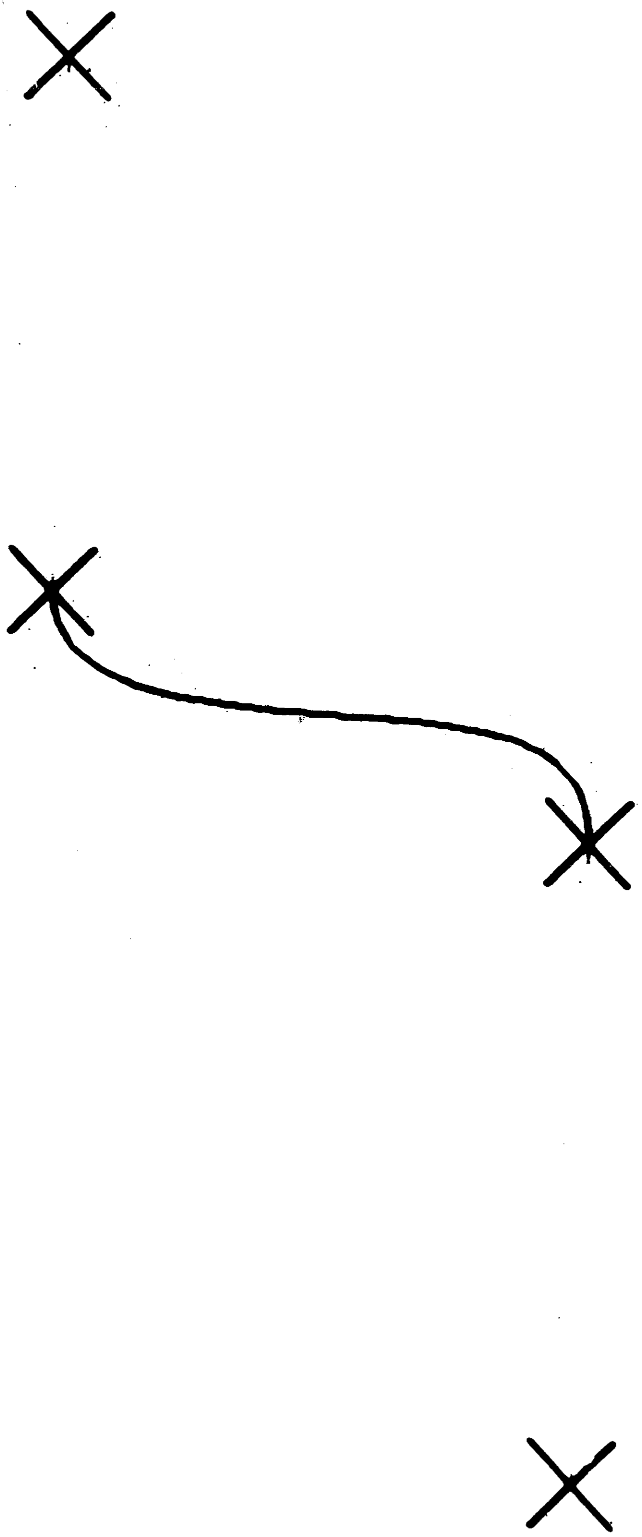


Figure 3—Curve with inflection point

or, in short, $B_{n+1} = T\,B_n$.

We have in mind placing the current values $\{P_n, \Delta P_n, \ldots, \Delta^m P_n\}$ (or the $\{\nabla^k P_n\}$) in a set of fast registers

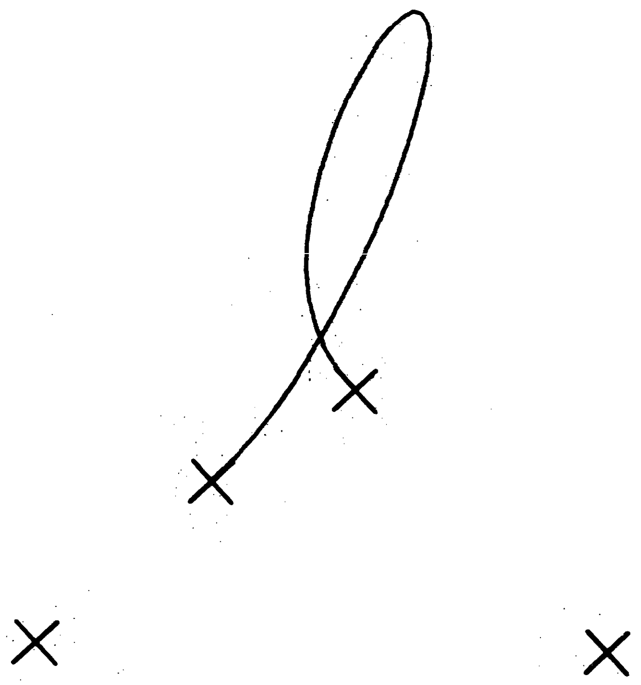


Figure 4—Curve which crosses itself

or memory cells. The successive values $\{P_n\}$ computed by either method will be used in some fashion to display the curve.

Let us factorize the matrices S and T to indicate the two computational schemes:

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_2\,A_1\,A_0$$

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_0\,A_1\,A_2$$

where A_i is the operation of adding $\Delta^{i+1}p$ to $\Delta^i p$ (or $\nabla^{i+1}p$ to $\nabla^i p$).

We perform these operations in the order A_0, A_1, A_2 for forward differences as indicated by the composition of operators $\mathbf{F}_{n+1} = A_2A_1A_0\mathbf{F}_n$ and in the opposite order for backward differences.

There are only three additions involved in either of these methods. However, there is a basic difference between them. In the forward difference scheme, each "new" value Δ^iP_{n+1} depends only on "old" values Δ^jP_n, but in the backward difference scheme each "new" value depends on the current value of the registers. It is therefore possible to execute all the additions for the forward difference scheme in parallel, taking only one addition time for evaluating successive values of the polynomials. This however requires much hardware (adders). The backward difference scheme is more economical for sequential computation in software, or in hardware with only one adder. By using extra logic one can however overlap the three additions in the same adder and execute them in not much more than one addition time.

Basic curve form

For our purposes a curve in R^N is a vector-valued function from the real line R to the real vector space R^N of dimension N.* That is, a curve is an ordered N-tuple of real functions $(f_1, f_2, \ldots, f_N)$. Furthermore, let us restrict the functions f_i to be linear combinations of some linearly independent set of functions, $\phi = [\phi_0, \phi_1, \ldots, \phi_M]$ and let the functions be defined over the domain [0, 1].

Let $f_j(u) = \sum_{i=0}^{M} A_{ij}\phi_i(u)$ where the $\{A_{ij}\}$ are constants. Then a curve segment can be represented by a unique $(M + 1) \times N$ real matrix A. A point $\mathbf{V}(u)$ on the curve associated with a parameter u is given by

$$\mathbf{V}(u) = [\phi_0(u), \phi_1(u), \ldots, \phi_M(u)]\, A = \phi(u)\, A =$$

$$= [f_1(u), f_2(u), \ldots, f_N(u)],\ 0 < u \leq 1.$$

In this context we will say either that the matrix A *represents* the curve, or, more pointedly, that the matrix A *is* the curve.

If we have some mapping $\mathbf{H}: R^N \rightarrow R^L$ from N dimensional into L dimensional space then the image $\mathbf{v}(u) = \mathbf{H}(\mathbf{V}(u))$ of the curve $\mathbf{V}$ under the mapping will be a curve $\mathbf{v}(u)$ in R^L. The matrix A does not uniquely represent the curve $\mathbf{v}(u)$. We can define the equivalence class $\mathbf{H}[A]$ of a matrix A under the mapping as the set of all matrices A_H such that $\mathbf{H}(\phi(u)A_H) = \mathbf{H}(\phi(u)A)$; it is clear that all the curves in R^N specified by the $A_H \epsilon\ \mathbf{H}[A]$ will be mapped into the same curve in R^L and there will thus be many representations of that curve. Notice that although the original function space—$[f_1, f_2, \ldots, f_N]$—is a vector space of rank not exceeding $(M + 1)$ its image under the mapping $\mathbf{H}$ is not necessarily of finite rank and we cannot in general represent the resulting curve in R^L by an expression of the form $\mathbf{W}(u) = \Psi(u)\ B$ where $\mathbf{W}(u) \epsilon R^L$, $\Psi(u) \epsilon R^P$ and set B a constant matrix of dimension $P \times L$, $\Psi(u)$ a fixed of basis functions, dependent only on $\mathbf{H}$ and ϕ. The mapping $\mathbf{H}$ will of course map the *curve* $[f_1, f_2, \ldots, f_N]$ into a *curve* $[g_1, g_2, \ldots, g_L]$ where $g_i = H_i[f_1, \ldots, f_N]$, where the $\{H_i\}$ are the components of the vector function $\mathbf{H}$.

We will usually choose the set of basis functions $[\phi_0, \ldots, \phi_M]$ to be the polynomials $[u^M, u^{M-1}, \ldots, u^2, u, 1]$ or some other set of linearly independent polynomials of degree M. The functions f_i will thus be the real polynomials of degree M or less. We will also choose N to be dimension 3 or 4 and L to be dimension 2 or 3 depending on whether we are talking about two or three dimensional curves. The mapping $\mathbf{H}$ will be that used in making the transformation from homogeneous coordinates to ordinary coordinates, namely, the projection obtained by dividing the first L components of the vector $\mathbf{V}$ (of dimension $L + 1 = N$) by the L + 1st component to arrive at the L components of the vector $\mathbf{H}(\mathbf{V})$. We thus identify the curve specified by the form ϕA and the resulting vector $\mathbf{V}(u)$ as *homogeneous coordinates* of a curve $\mathbf{v}(u)$ in dimension one less. The many-to-one property of the mapping $\mathbf{H}$ is illustrated by the fact that two points in homogeneous coordinates, $\mathbf{P}$ and $\mathbf{Q}$, are equivalent if and only if $\mathbf{P} = \alpha\mathbf{Q}$ for some non-zero α.

Thus we talk about curves of the form

$$x = \frac{a_Mu^M + a_{M-1}u^{M-1} + \ldots + a_1u + a_0}{d_Mu^M + d_{M-1}u^{M-1} + \ldots + d_1u + d_0}$$

$$y = \frac{b_Mu^M + b_{M-1}u^{M-1} + \ldots + b_1u + b_0}{d_Mu^M + d_{M-1}u^{M-1} + \ldots + d_1u + d_0}$$

and optionally,

$$z = \frac{c_Mu^M + c_{M-1}u^{M-1} + \ldots + c_1u + c_0}{d_Mu^M + d_{M-1}u^{M-1} + \ldots + d_1u + d_0}$$

The value for each component of a point on the curve is the ratio of two polynomials in the parameter u. In theory and in practice we will *first* do some operations on the curve in R^N (or upon its representation as a matrix) and only at the last moment before display

* In other words, we only consider curves in a parametric (explicit), rather than implicit, form.

perform the mapping $H : R^N \rightarrow R^L$ by doing the division to remove the homogeneous coordinate. In most cases the many-to-one property of **H** will not come into play explicitly but only when we remember that the points $V(u)$ are expressed in homogeneous coordinates.

If we have two sets of basis functions for the original function space $[f_1, \ldots, f_N]$, say $[\phi_0, \phi_1, \ldots \phi_M]$ and $[\Psi_0, \Psi_1, \ldots \Psi_M]$ we can always write the curve in two forms.

$V(u) = [\phi_0, \phi_1, \ldots, \phi_M]\, A = [\Psi_0, \Psi_1, \ldots, \Psi_M]\, B$, $B = T\,A$ where T is the change of basis transformation, $[\Psi_0, \Psi_1, \ldots, \Psi_M]\, T = [\phi_0, \phi_1, \ldots, \phi_M]$. This relation will prove extremely useful in the practical matter of determining the matrix A to represent a curve under the basis $\phi = [\phi_0, \phi_1, \ldots, \phi_M]$ where ϕ will be chosen so as to make the display easy where a different basis $[\Psi_0, \ldots, \Psi_M]$ might be more convenient for specifying the curve or performing computations upon it.

Reparameterization and shape invariance

Supposing we have a curve represented by the matrix A, it is natural to ask what matrices B represent the same curve. The major reason for investigating this shape invariance is to enable us to draw a particular curve in as smooth a manner as possible, while using equally spaced $\{t_i\}$. The initial choice of a matrix to represent a curve may be poorly made, as when the velocity vector $\frac{dv}{du}$ is great at the same time the curvature of the curve is great.* We prefer to have the velocity slow at such points in order that successive points in time along the curve well represent its shape. (See Figure 5).

We define two curves in R^L to be the same if and only if there exists a one-to-one correspondence between the points on the curves such that for each point $\mathbf{p}(s)$ on curve A there exists a point $\mathbf{q}(u)$ on curve B such that $\mathbf{p}(s) = \mathbf{q}(u)$. This correspondence between the points on the curves induces a correspondence between the parameters on the curves, which we will denote by a function s such that $s = s(u)$, or $\mathbf{p}(s) = \mathbf{p}(s(u)) = \mathbf{q}(u)$. In this context we may have to extend the domain of one or the other of the curves to cover the entire real line, $-\infty < u < +\infty$, $-\infty < s < +\infty$, although we will still only display a finite portion, $V(u) = \phi(u)\, B$, and $0 \leqq u \leqq 1$. (See Figure 6).

In the notation of the previous section, we have

$$\mathbf{H}\{[s^M\ s^{M-1} \ldots s\ 1]\, A\} = \mathbf{H}\{[u^M\ u^{M-1} \ldots u\ 1]\, B\}$$

* We normally think of the parameter u as being time, and derivatives with respect to it as velocities, for reasons to become obvious later.

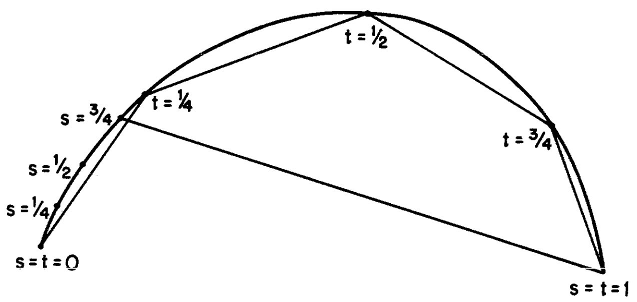


Figure 5—Good and bad parameterizations

where **H** is the operation of performing the homogeneous division. This equality between the coordinates on the curves will hold at the point $\mathbf{p}(s(u)) = \mathbf{q}(u)$ if and only if there exists a non-zero scalar factor α associated with the point $\mathbf{p}(s(u)) = \mathbf{q}(u)$ and hence a function $\alpha(u)$ of the parameter u such that

$$\alpha(u)\, [s^M\ s^{M-1} \ldots s\ 1]\, A = [u^M\ u^{M-1} \ldots u\ 1]\, B\,.$$

Write the above equation for $M + 1$ different values of u, and get:

$$\begin{bmatrix} \alpha(u_0) & & & \\ & \alpha(u_1) & & \\ & & & \\ & & & \alpha(u_M) \end{bmatrix} \begin{bmatrix} s(u_0)^M & s(u_0)^{M-1} & \ldots\ s(u_0) & 1 \\ s(u_1)^M & s(u_1)^{M-1} & s(u_1) & 1 \\ \vdots & \vdots & \vdots & \vdots \\ s(u_M)^M & s(u_M)^{M-1} & s(u_M) & 1 \end{bmatrix} A = $$

$$= \begin{bmatrix} u_0^M & u_0^{M-1} & \ldots & 1 \\ u_1^M & u_1^{M-1} & \ldots & 1 \\ \vdots & \vdots & & \vdots \\ u_M^M & u_M^{M-1} & & 1 \end{bmatrix} B$$

The matrix which precedes the B matrix is a Vandermonde matrix,[6] which is non-singular since the $\{u_i\}$ are all different. Note that the matrix preceding the A is also a non-singular Vandermonde matrix, as different $\{u_i\}$ imply different $\{s_i\}$.

Solve for B:

$$B = \begin{bmatrix} u_0^M & \ldots & 1 \\ \vdots & & \vdots \\ u_M^M & \ldots & 1 \end{bmatrix}^{-1} \begin{bmatrix} \alpha(u_0) & \\ & \alpha(u_M) \end{bmatrix}$$

$$\begin{bmatrix} s(u_0)^M & \ldots & 1 \\ \vdots & & \vdots \\ s(u_M)^M & \ldots & 1 \end{bmatrix} A$$

which we write as B = S A for some as yet undetermined square $(M + 1) \times (M + 1)$ matrix S. Since S is the product of three non-singular matrices, it is non-singular.

The original equation becomes

$$\alpha(u)\ [s^M\ s^{M-1} \ldots 1]\ A = [u^M\ u^{M-1} \ldots 1]\ S\ A$$

which holds if:

$$\alpha(u)\ [s^M\ s^{M-1} \ldots 1] = [u^M\ u^{M-1} \ldots 1]\ S\ .$$

If the rank of the null space of A is zero then it will be a necessary condition as well. We will not at this time pursue a further characterization of curves in terms of the null space of their associated matrix.

Under these conditions equating appropriate components of the vectors in the above equation gives:

1. for the last (M + 1st) component,

$$\alpha(u)\ 1 = [u^M \ldots 1]\ S \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = P(u)$$

P is a polynomial of degree not greater than M. If $P = \alpha(u)$ is a constant then s = u and S = kI for some constant k, a trivial and uninteresting equality of curves. Hence we assume P is not a constant.

2. for the Mth component:

$$\alpha(u)\ s = [u^M \ldots 1]\ S \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix} = Q(u)$$

Q is a polynomial of degree not exceeding M and $s = Q/\alpha(u) = Q(u)/P(u)$

3. for the first component:

$$\alpha(u)\ s^M = [u^M \ldots 1]\ S \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = R(u)$$

R is a polynomial of degree not exceeding M and

$$s^M = \frac{R}{\alpha(u)} = \frac{R}{P} = \left[\frac{Q}{P}\right]^M$$

The expression $R/P = [Q/P]^M$ has at most M poles

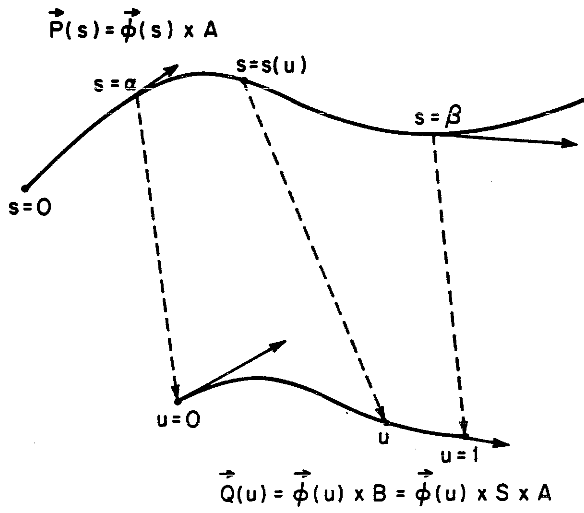


Figure 6—Shape invariant transformation

and at most M zeros (including multiplicities) since the polynomials R and P are of degree not greater than M. Thus Q/P has at most one zero and one pole and we have

$$s = \frac{Q}{P} = \frac{au + b}{cu + d}\,.$$

Substituting in the original equation gives:

$$\alpha(u) \left[\left(\frac{au+b}{cu+d}\right)^M \left(\frac{au+b}{cu+d}\right)^{M-1} \ldots 1\right]$$
$$= [u^M \ldots 1]\ S =$$

$$= \frac{\alpha(u)}{(cu+d)^M} [[(au + b)^M], [(au + b)^{M-1}(cu + d)], \ldots, [(cu + d)^M]]$$

Each component of $[u^M \ldots 1]$ S is a polynomial of degree not exceeding M in u. The components on the right hand side above must thus be the same polynomials. Hence $\alpha(u) = e(cu + d)^M$, where e is a constant, and by inspection the component S_{ij} of the matrix S is the coefficient of u^{M-i} in the expansion of $e(au + b)^{M-j}(cu + d)^j$.

Bypassing the algebra,

$$S_{ij} = e \sum_{k=0}^{j} a^{M-i-j+k}\, b^{i-k}\, c^{j-k}\, d^k \binom{M-j}{M-i-j+k}\binom{j}{k}$$

where we use the convention that $\binom{j}{k} = 0$ when j, k are outside of the range $0 \leq k \leq j$.

The matrix S for M = 2 is

$$\begin{bmatrix} a^2 & ac & c^2 \\ 2ab & bc + ad & 2cd \\ b^2 & bd & d^2 \end{bmatrix}$$

and the matrix for M = 3 is

$$\begin{bmatrix} a^3 & a^2c & ac^2 & c^3 \\ 3a^2b & 2abc + a^2d & bc^2 + 2acd & 3c^2d \\ 3ab^2 & b^2c + 2abd & 2bcd + ad^2 & 3cd^2 \\ b^3 & b^2d & bd^2 & d^3 \end{bmatrix}$$

Typically we will want to use the reparameterization matrix S to change the amount (extent) of a given curve that is drawn and to alter the rate at which the moving vector $[u^M \ldots 1]$ S A traverses the curve. These variables can be specified by setting

1. $s(0) = \alpha$
2. $s(1) = \beta$
3. $\frac{s'(0)}{s'(1)} = r^2$

α is the value of the parameter s on the original curve A at which the new curve B begins. β is the value of the parameter s on A at which the curve B ends. r^2 is a ratio of derivatives; the significance of r will be shown shortly. It is easily verified that the ratio

$$\frac{s'(u_1)}{s'(u_2)} = \left(\frac{cu_2 + d}{cu_1 + d}\right)^2 \geqq 0$$

Substituting the above conditions into $s = (au + b)/(cu + d)$ we solve for a, b, c, d:

$$a = \beta - \alpha r$$
$$b = \alpha r \qquad \text{or } s = \frac{(\beta - \alpha r)u + \alpha r}{(1 - r)u + r}$$
$$c = 1 - r$$
$$d = r$$

using the arbitrary condition c + d = 1 to remove the extra degree of freedom introduced by the homogeneous form of the function s. The parameter e will allow an absolute adjustment of the homogeneous coordinate system, if that is desired.

The velocity vector $d\mathbf{v}/ds$ at the beginning ($u = 0$, $s = \alpha$) will be multiplied by the factor $(\beta - \alpha)/r$ and at the end ($u = 1$, $s = \beta$) by $(\beta - \alpha)r$.

In fact, in general we have

$$\frac{dp}{du} = \frac{ds}{du}\frac{dp}{ds} = \frac{(\beta - \alpha)r}{[(1 - r)u + r]^2}\frac{dp}{ds}$$

The product of the magnitudes of the velocity vectors at the ends of the new curve B is thus $(\beta - \alpha)^2$ times the product of the magnitudes of the velocity vectors at the corresponding points $s = \alpha$, $s = \beta$ of the original curve A. This product thus remains constant for all similar curves covering the same interval (α, β) of A. It is in any case independent of the rate parameter r. Notice that if $r < 0$ the new curve will not completely overlap the old curve in the range $0 \leqq u \leqq 1, 0 \leqq s \leqq 1$ although over the full line $-\infty < u < +\infty$, $-\infty < s < +\infty$ the two curves will be identical. In particular, at the value $u = r/(r - 1)$, $s(u) = \pm\infty$, definitely outside the domain of the original curve $\mathbf{v}(s)$. Thus, for example, if the matrix A draws half a circle, the matrix S A will draw the other half in the same direction if $\alpha = 1, \beta = 0, r = -1$. (See Figure 7 for an example of the use of a continuation matrix.)

For M = 2 this matrix s(1, 0, −1) is

$$\begin{bmatrix} 1 & 2 & 4 \\ -2 & -3 & -4 \\ 1 & 1 & 1 \end{bmatrix}$$

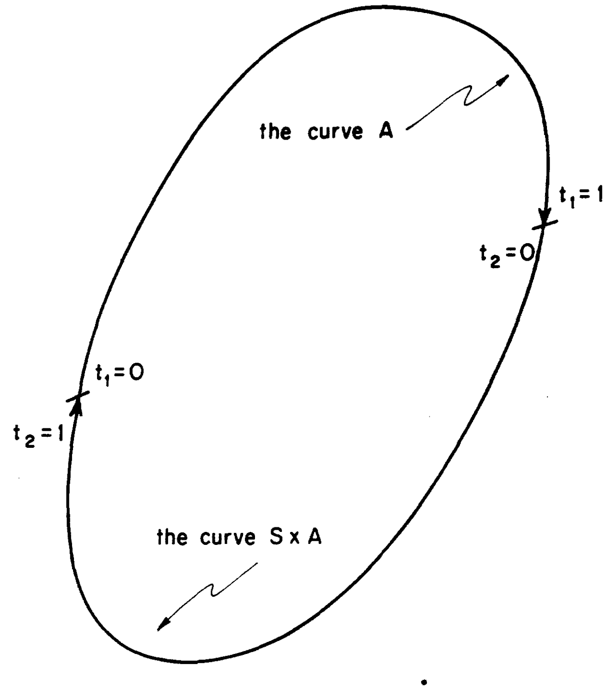


Figure 7—Continuation of a curve

and for M = 3 it is

$$\begin{bmatrix} 1 & 2 & 4 & 8 \\ -3 & -3 & -8 & -12 \\ 3 & 4 & 5 & 6 \\ -1 & -1 & -1 & -1 \end{bmatrix}$$

If we have a curve $\mathbf{v}(u) = \mathbf{H}[\Psi\ A_T]$ represented by a matrix A_T for some other basis function $\Psi = [u^M\ u^{M-1} \ldots 1]\ T$ then the reparameterization matrix S_T for this basis will be given by the similarity transformation $S_T = T^{-1}\ S\ T$ and the curve matrix after reparameterization will be $T^{-1}\ S\ T\ A_T$.

Endpoint-Derivative form

In the previous sections we have been talking about curves of arbitrary degree; in this section we will be concerned only with curves of degree m = 3. Let $h\mathbf{v} = \mathbf{V}$ denote the homogeneous coordinates of a point in R^N, that is, let $h = V_N$, $\mathbf{v} = \mathbf{V}/V_N = \mathbf{V}/h$ where $\mathbf{v}$ is a vector of the form $[v_1\ v_2 \ldots v_{N-1}\ 1]$. Then the equation of a curve is given parametrically by

$$\mathbf{V}(u) = h(u)\ \mathbf{v}(u) = \phi(u)A$$

where $\phi(u)$ are appropriate M dimensional basis functions. The curve in $R^{N-1} = R^L$ is obtained by taking the first L components of the vector $\mathbf{v}(u)$; that is, by dividing out the homogeneous coordinate h(u) from the vector $h(u)\mathbf{v}(u)$ and dropping the last component.

Let

$$\mathbf{V}(0) = \mathbf{V}_0$$

$$\mathbf{V}(1) = \mathbf{V}_1$$

$$\mathbf{V}'(0) = \left.\frac{d\mathbf{V}(u)}{du}\right|_{u=0} = \mathbf{V}_0'$$

$$\mathbf{V}'(1) = \mathbf{V}_1$$

then we have

$$\begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_0' \\ \mathbf{V}_1' \end{bmatrix} = \begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi'(0) \\ \phi'(1) \end{bmatrix} A$$

One can show that the square matrix

$$\begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi'(0) \\ \phi'(1) \end{bmatrix}$$

is non-singular since the basis functions $\phi(u)$ are by definition four linearly independent polynomials of degree 3. Define

$$M = \begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi'(0) \\ \phi'(1) \end{bmatrix}$$

$$A = M \begin{bmatrix} \mathbf{V}_0 \\ \mathbf{V}_1 \\ \mathbf{V}_0' \\ \mathbf{V}_1' \end{bmatrix}$$

We now write

$$\mathbf{V}_0 = h_0\ \mathbf{v}_0$$

$$\mathbf{V}_1 = h_1\ \mathbf{v}_1$$

$$\mathbf{V}_0' = \left.\frac{d(h\mathbf{v})}{du}\right|_{u=0} = h_0'\ \mathbf{v}_0 + h_0\ \mathbf{v}_0'$$

$$\mathbf{V}_1 = h_1'\ \mathbf{v}_1 + h_1\ \mathbf{v}_1'$$

where $\mathbf{v}_0'$, $\mathbf{v}_1'$ are the derivatives with respect to the parameter u and where h_0, h_1, h_0', h_1' are meaningful only in homogeneous coordinates. Notice that the last component of $\mathbf{v}_0'$ and $\mathbf{v}_1'$ is 0.

Thus

$$A = M \begin{bmatrix} h_0 & 0 & 0 & 0 \\ 0 & h_1 & 0 & 0 \\ h_0' & 0 & h_0 & 0 \\ 0 & h_1' & 0 & h_1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix}$$

The curve is given by

$$\phi(u)\ A = \phi(u)\ M \begin{bmatrix} h_0 & & & \\ & h_1 & & \\ h_0' & & h_0 & \\ & h_1' & & h_1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix}$$

We can perform the multiplication $\phi(u)$ M separately to define a new set of basis polynomials $[F_0\ F_1\ G_0\ G_1] = \phi$ M with transformation of basis matrix M itself.* We observe that

* This notation is borrowed freely from S. A. Coons.[7,8]

$$\begin{bmatrix} F_0(0) & F_1(0) & G_0(0) & G_1(0) \\ F_0(1) & F_1(1) & G_0(1) & G_1(1) \\ F_0'(0) & F_1'(0) & G_0'(0) & G_1'(0) \\ F_0'(1) & F_1'(1) & G_0'(1) & G_1'(1) \end{bmatrix} = \begin{bmatrix} \phi(0) \\ \phi(1) \\ \phi'(0) \\ \phi'(1) \end{bmatrix} M =$$

$$= M^{-1}M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With these basis functions, a curve can *always* be represented by a matrix A_M of the form

$$A_M = \begin{bmatrix} h_0 & & & \\ & h_1 & & \\ h_0 & & h_0 & \\ & h_1 & & h_1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix}$$

since it is given by the equation

$$\mathbf{V}(u) = [F_0\ F_1\ G_0\ G_1] \begin{bmatrix} h_0 & & & \\ & h_1 & & \\ h_0 & & h_0 & \\ & h_1 & & h_1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix}$$

We will sometimes write this endpoint-derivative form as

$$\mathbf{V}(u) = [F_0\ F_1\ G_0\ G_1] \begin{bmatrix} (h\mathbf{v})_0 \\ (h\mathbf{v})_1 \\ (h\mathbf{v})_0' \\ (h\mathbf{v})_1' \end{bmatrix}$$

It will also prove useful to rewrite it in the form

$$\mathbf{V}(u) = [h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0 & & G_0 & \\ & F_1 & & G_1 \\ G_0 & & & \\ & G_1 & & \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix}$$

or in standard form as

$$V(u) = [(F_0h_0 + G_0h_0'), (F_1h_1 + G_1h_1'), (G_0h_0), (G_1h_1)] \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix}$$

where now the matrix

$$A = \begin{bmatrix} v_0 \\ v_1 \\ v_0 \\ v_1 \end{bmatrix}$$

is especially simple, although for a very special basis function constructed specifically for the particular set of homogeneous coordinates h_0, h_1, h_0', h_1'. This form shows that a point on the curve $\mathbf{V}(u)$ is always a linear combination of the vectors $\mathbf{v}_0$, $\mathbf{v}_1$, $\mathbf{v}_0'$, $\mathbf{v}_1'$.

Curve specification

In the previous section we have characterized a rational parametric cubic polynomial curve in terms of endpoints—$\mathbf{v}_0'$, $\mathbf{v}_1'$—and tangent vectors at the endpoints—$\mathbf{v}_0$, $\mathbf{v}_1$—with four remaining degrees of freedom, h_0, h_1, h_0', h_1'. In this section we investigate several ways of computing the numbers h_0, h_1, h_0', h_1' in order to completely specify the curve from geometric consideration. (See Figure 8). Although some or all of these results may be inappropriate for a particular application the techniques used indicate the generality of the curve formulation and what we believe to be the proper way to attack the problem.

Let us require the curve to pass through the point $\mathbf{v}_c$ at the value u_c of the parameter. We have

$$\mathbf{v}_c = [h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0(u_c) & 0 & G_0(u_c) & 0 \\ 0 & F_1(u_c) & 0 & G_1(u_c) \\ G_0(u_c) & 0 & 0 & 0 \\ 0 & G_1(u_c) & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \mathbf{v}_0' \\ \mathbf{v}_1' \end{bmatrix} = \mathbf{h}\,\mathfrak{F}\,\boldsymbol{\nu}$$

We can solve $\mathbf{v}_c = \mathbf{h}\mathfrak{F}\boldsymbol{\nu}$ for $\mathbf{h}\mathfrak{F}$ (and hence for $\mathbf{h}$, since $\mathcal{f}$ is non-singular for any $0 < u_c < 1$) if and only if $\mathbf{v}_c$ belongs to the range of ν. In R^3 the geometrical meaning of this condition is that if the curve is planar, $\mathbf{v}_c$ must be in the same plane, or, if the curve is linear, $\mathbf{v}_c$ must be on the same line. If the curve is really three-dimensional then $\boldsymbol{\nu}$ will have full rank and we can solve directly for

$$\mathbf{h} = \mathbf{v}_c\,\boldsymbol{\nu}^{-1}\,\mathfrak{F}^{-1} = \mathbf{v}_c\,(\mathfrak{F}\boldsymbol{\nu})^{-1}$$

In R^2 the equation $\mathbf{v}_c = \mathbf{h}\mathfrak{F}\boldsymbol{\nu}$ is indeterminate since we have three equations and four unknowns. If $\mathbf{v}_c$ is in the range of $\boldsymbol{\nu}$ we can impose an additional condition.* One convenient such condition is to specify that the slope at $\mathbf{v}_c$ be given by the ratio $x'_c/y'_c = dx/dy$.

* $\mathbf{v}_c$ will not be in the range of $\boldsymbol{\nu}$ if and only if $\mathbf{v}_0'$, $\mathbf{v}_1'$ and $\mathbf{v}_0 - \mathbf{v}_1$ are collinear and $\mathbf{v}_c - \mathbf{v}_1$ is not on the same line. This means that the tangents at each endpoint point to the other end *and* $\mathbf{v}_c$ is not on the line between the two endpoints.

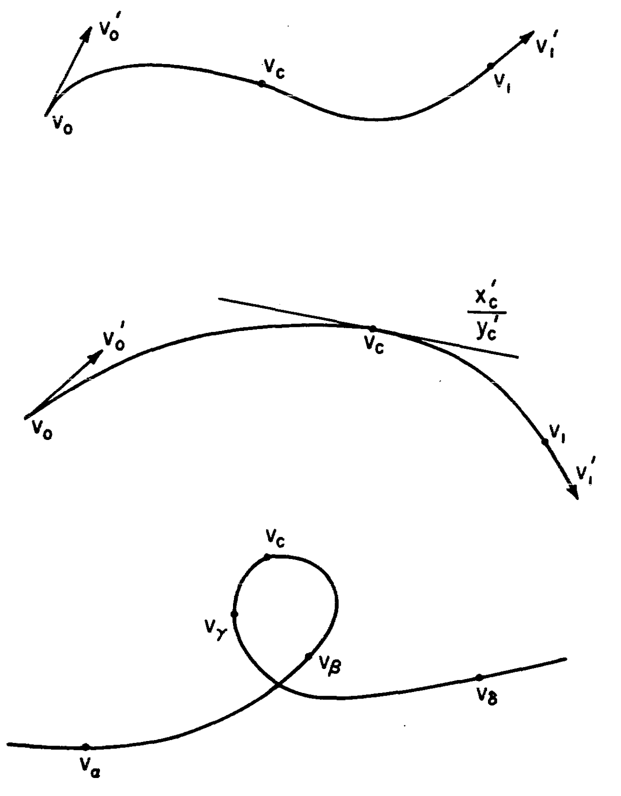


Figure 8—Specifying the curve

Taking derivatives with respect to u we have

$$[h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0' & & G_0' & \\ & F_1' & & G_1' \\ G_0' & & & \\ & G_1' & & \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_0' \\ v_1' \end{bmatrix} =$$

$$= \left.\frac{d(h_c v_c)}{du}\right|_{u\,=\,u_c} = h_c' v_c + h_c v_c' = h_c' v_c + v_c'$$

because we have let $h_c = 1$. ($v_c = [x_c\ y_c\ 1]$). Since the third component of v_c', v_0', v_1' is 0 we can solve for

$$h_c' = [h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0' \\ F_1' \\ G_0' \\ G_1' \end{bmatrix}$$

and thus

$$[h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0' & & G_0' & \\ & F_1' & & G_1' \\ G_0' & & & \\ & G_1' & & \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_0' \\ v_1' \end{bmatrix} =$$

$$= [h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0' \\ F_1' \\ G_0' \\ G_1' \end{bmatrix} v_c + v_c'$$

or, rearranging,

$$[h_0\ h_1\ h_0'\ h_1'] \begin{bmatrix} F_0' & & G_0' & \\ & F_1' & & G_1' \\ G_0' & & & \\ & G_1' & & \end{bmatrix} \begin{bmatrix} v_0 - v_c \\ v_1 - v_c \\ v_0' \\ v_1' \end{bmatrix} =$$

$$= v_c' = [x_c'\ y_c'\ 0]$$

Denote the left side of the above by Q. Since we are only concerned with the slope at v_c, we can extract the first and second components by a post-multiplication and obtain the ratio

$$\frac{Q\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}{Q\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}} = \frac{x_c'}{y_c'} \quad \text{whence} \quad Q\begin{bmatrix} y_c' \\ -x_c' \\ 0 \end{bmatrix} = 0$$

If we adjoin this single equation to the original equation for the curve passing through the point v_c we have

$$[h_0\ h_1\ h_0'\ h_1'] \left[\begin{bmatrix} F_0 & & G_0 & \\ & F_1 & & G_1 \\ G_0 & & & \\ & G_1 & & \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_0' \\ v_1' \end{bmatrix} \right|$$

$$\left| \begin{bmatrix} F_0' & & G_0' & \\ & F_1' & & G_1' \\ G_0' & & & \\ & G_1' & & \end{bmatrix} \begin{bmatrix} v_0 - v_c \\ v_1 - v_c \\ v_0' \\ v_1' \end{bmatrix} \begin{bmatrix} y_c' \\ -x_c' \\ 0 \end{bmatrix} \right] = [v_c|0]$$

If the matrix is invertible, we can solve for h_0, h_1, h_0', h_1' and determine a curve passing through a specified point v_c with a given slope x_c'/y_c' with given endpoints v_0, v_1 and tangent vectors v_0', v_1' at those endpoints. As with the three dimensional case above we could characterize this problem in somewhat more generality in terms of the range of the appropriate matrices, but we will not do so since the conditions are not simple enough to be interesting.

We could in the previous examples have just as easily required the curve to pass through four non-planar points v_α, v_β, v_γ, v_δ at different values α, β, γ, δ of the parameter.

We would have

$$\begin{bmatrix} \phi(\alpha) \\ \phi(\beta) \\ \phi(\gamma) \\ \phi(\delta) \end{bmatrix} A = \begin{bmatrix} h_\alpha \mathbf{v}_\alpha \\ h_\beta \mathbf{v}_\beta \\ h_\gamma \mathbf{v}_\gamma \\ h_\delta \mathbf{v}_\delta \end{bmatrix}$$

or

$$A = \begin{bmatrix} \phi(\alpha) \\ \phi(\beta) \\ \phi(\gamma) \\ \phi(\delta) \end{bmatrix}^{-1} \begin{bmatrix} h_\alpha \mathbf{v}_\alpha \\ h_\beta \mathbf{v}_\beta \\ h_\gamma \mathbf{v}_\gamma \\ h_\delta \mathbf{v}_\delta \end{bmatrix}$$

Then let us ask that the curve pass through $\mathbf{v}_c$ at $u = u_c$:

$$\phi(u_c)\, M \begin{bmatrix} h_\alpha \mathbf{v}_\alpha \\ h_\beta \mathbf{v}_\beta \\ h_\gamma \mathbf{v}_\gamma \\ h_\delta \mathbf{v}_\delta \end{bmatrix} = \mathbf{v}_c$$

where now we have let

$$M = \begin{bmatrix} \phi(\alpha) \\ \phi(\beta) \\ \phi(\gamma) \\ \phi(\delta) \end{bmatrix}$$

If we define $\phi(u_c)\, M = [F_0\ F_1\ F_2\ F_3]$ we have after rearranging

$$[h_\alpha\ h_\beta\ h_\gamma\ h_\delta] \begin{bmatrix} F_0\ \mathbf{v}_\alpha \\ F_1\ \mathbf{v}_\beta \\ F_2\ \mathbf{v}_\gamma \\ F_3\ \mathbf{v}_\delta \end{bmatrix} = \mathbf{v}_c$$

which can be solved directly for $[h_\alpha\ h_\beta\ h_\gamma\ h_\delta]$ if the matrix is non-singular, that is, if the four points are non-planar. In this way we have asked the three-dimensional curve to pass through five specific points at five specific values of the parameter. Similar results could be derived in two dimensions.

REFERENCES

1 R F SPROULL I E SUTHERLAND
A clipping divider
Proc F J C C 1968

2 I E SUTHERLAND
A head-mounted three dimensional display
Proc F J C C 1968

3 L G ROBERTS
Homogeneous matrix representation and manipulation of n-dimensional constructs
The Computer Display Review Adams Associates May 1965

4 H F BAKER
Principles of geometry
6 Vol., Cambridge University Press Cambridge 1922 +

5 R M WINGER
An introduction to projective geometry
D C Heath and Co Boston 1923

6 G BIRKHOFF S MACLANE
A survey of modern algebra
Macmillan New York 1965

7 S A COONS B HERZOG
Surfaces for computer-aided aircraft design
Presented at AIAA 4th Annual Meeting and Technica Display Anaheim California October 1967 American Inst Aeronautics and Astronautics, New York

8 S A COONS
Surfaces for computer-aided design of space forms
Project MAC Report MAC–TR–41 MIT June 1967

A class of surfaces for computer display*

by THEODORE M. P. LEE**

Harvard University
Cambridge, Massachusetts

INTRODUCTION

This paper describes the mathematical formulation of a class of three-dimensional surfaces parametrically represented for efficient computer display. The degrees of freedom in the representation are such as to provide a rich variety of surfaces with convenient parameters for manipulation and constraint satisfaction. Historically this work began as an investigation of the properties of rational parametric cubics, a class of curves well-suited to the Harvard three-dimensional display.[1,2] The desire to represent curvilinear surfaces in terms of these curves and an introduction to the Coons' surface formulation[3] were sufficient to suggest the approach discussed here.

The particular advantages of this approach with respect to projective transformations and rapid iterative display did not become apparent until later, although they may be its most attractive features. The ability to truly and simply represent such classic surfaces as the sphere and torus, although a desired goal, was not demonstrated until even later.

The paper begins with an introduction to homogeneous coordinate geometry, a topic now out of favor in the general college curriculum. I apologize to those who may have seen this material before, but it is necessary for a proper understanding of the results presented, especially those dealing with continuity conditions. The rest of the paper contains primarily mathematical techniques for manipulating the surfaces.

Notation

We will be talking about surfaces, represented as tensors, curves, represented as matrices, or points, represented as either three dimensional (ordinary coordinates) or four dimensional (homogeneous coordinates) vectors. A vector, usually a point in homogeneous coordinates, will always be denoted by boldface type, for example, **V**. Where relevant, a four dimensional vector will be represented by an upper-case letter and a three dimensional vector by a lower-case letter. Points or curves may be obtained as part of a higher order entity or as separate entities.

Subscripts will be used to denote either components of the array (tensor, matrix, or vector) or to indicate partial derivatives with respect to the parameters. Components of vectors will *not* be in boldface although vector components of a matrix will, for example, the vector $\mathbf{A}_i$ of the matrix A_{ij}. Integer subscripts—$_i$, $_j$, $_k$—will be used for components in some cases while the symbols $_{hx}$, $_{hy}$, $_{hz}$, $_h$ will be used when it is desirable to emphasize the spatial coordinates. The subscripts $_u$, $_v$ will naturally refer to the partial derivatives $\frac{\partial}{\partial u}$, $\frac{\partial}{\partial v}$. The components of a point in three dimensions are indicated by subscripts $_x$, $_y$, $_z$—for example, p_x. This implies either true three dimensional data or a division to remove the homogeneous scale factor.

When a tensor describing a surface is used *without* any subscripts for components it is to be treated not as an array of scalars but rather as a matrix whose elements are vectors. Multiplication of such a matrix of vectors by a matrix of scalars is to be interpreted in the standard way, with the summation being a vector sum

* The research reported here was performed at Harvard University and supported in part by the Advanced Research Projects Agency (ARPA) of the Department of Defense under contract SD-265 with Harvard University, under contract AF 30(602)-4277 with the University of Utah, by the Office of Naval Research under contract ONR 1866(16) with Harvard University, and by a long standing agreement between Bell Telephone Laboratories and the Harvard Computation Laboratory. The author is a National Science Foundation graduate fellow.

** At the Aiken Computation Laboratory, of the Division of Engineering and Applied Physics.

and the individual multiplications being the product of a scalar and a vector.

When convenient, the standard convention of summing over repeated indices will be used, with the proviso that indices appearing on both sides of an equation will not be summed but indicate a running index. Such summation and running indices will take on the values 0, 1, 2, 3 for virtually all of the paper.

A curve, S(u, v), u = a, a = constant, $0 \leq v \leq 1$, on a surface S(u, v) will be denoted by $S^{u=a}$ or by S^{av} for brevity. A point, S(u, v), u = a, v = b, on the surface will be denoted by $S^{u=a,\ v=b}$ or by S^{ab}. In most cases it does not matter whether such notation is taken to represent the object (surface, curve, or point) itself or the array describing the object; context should make the situation clear. The geometric and mathematical bases in which such an array represents an object will either be irrelevant or clearly specified.

Homogeneous coordinates

Taking the lead from the 19th century projective geometrists,[4] computer display programmers have recently adopted the use of a *homogeneous coordinate* representation of geometrical data.[5] This section summarizes a few of the definitions, conventions and formulae in homogeneous analytic projective geometry needed for an understanding of the surface formulation. Most of this material has been printed elsewhere, but is not readily available.[6,7]

Point: A point (in three dimensions) is represented by a non-zero vector of four components. Two points **P**, **Q** are to be regarded as the same point if and only if they are linearly dependent; that is, if and only if there exists a non-zero constant α such that $\mathbf{P} = \alpha\mathbf{Q}$. The ordinary three dimensional point [x, y, z] can and will be represented in the homogeneous form as [x, y, z, 1]; hence any point [a, b, c, d], d ≠ 0, is equivalent to the three dimensional point [a/d, b/d, c/d]. This representation will be indicated by the notation [hx, hy, hz, h] = h**v** = h[x y z 1], **v** = [x y z 1] for the homogeneous coordinates of a point V, where by implication we take the quotients hx/h, hy/h, hz/h to find the three dimensional coordinates. The assumption is that once a point—[hx, hy, hz, h]—has been obtained, something—hardware or software—will perform the necessary division by the homogeneous coordinate. In particular, whenever we talk about a point to be displayed, this division must be performed.

Line: Three points **P**, **Q**, **R** are collinear if and only if they are linearly dependent. The set of points **P** on the line through two points **Q**, **R** can thus be generated parametrically by $\mathbf{P} = \alpha\mathbf{Q} + (1 - \alpha)\mathbf{R}$, or, in full generality, by $\mathbf{P} = \alpha\mathbf{Q} + \beta\mathbf{R}$, α, β not both zero. Two points on this line, $\mathbf{P}_1 = \alpha_1\mathbf{Q} + \beta_1\mathbf{R}$ and $\mathbf{P}_2 = \alpha_2\mathbf{Q} + \beta_2\mathbf{R}$ are equivalent if and only if the vectors $[\alpha_1, \beta_1]$, $[\alpha_2, \beta_2]$ are linearly dependent, that is, in this case, proportional.

Plane: Four points **P**, **Q**, **R**, **S** are coplanar if and only if they are linearly dependent. Hence the plane through three points **P**, **Q**, **R** is spanned by $\alpha\mathbf{P} + \beta\mathbf{Q} + \gamma\mathbf{R}$, α, β, γ not all zero. Furthermore, for the dependence $\alpha\mathbf{P} + \beta\mathbf{Q} + \gamma\mathbf{R} + \delta\mathbf{S} = 0$ to hold, we must have $[\mathbf{P\ Q\ R\ S}] = 0$, or, expanding by minors on the fourth column, $\mathbf{S}\cdot\mathbf{T} = 0$ (vector dot product) where

$$\mathbf{T} = \left[\begin{vmatrix} P_2 & Q_2 & R_2 \\ P_3 & Q_3 & R_3 \\ P_4 & Q_4 & P_4 \end{vmatrix}, \quad -\begin{vmatrix} P_1 & Q_1 & R_1 \\ P_3 & Q_3 & R_3 \\ P_4 & Q_4 & R_4 \end{vmatrix}, \begin{vmatrix} P_1 & Q_1 & R_1 \\ P_2 & Q_2 & R_2 \\ P_4 & Q_4 & R_4 \end{vmatrix}, \quad -\begin{vmatrix} P_1 & Q_1 & R_1 \\ P_2 & Q_2 & R_2 \\ P_3 & Q_3 & R_3 \end{vmatrix}\right]$$

The relation $\mathbf{S}\cdot\mathbf{T} = 0$ is thus the equation of a plane and the vector **T** represents the plane. Among many results it can be shown that two planes, **T**, **U** are equivalent if and only if the vectors **T**, **U** are proportional.

Transformations

In ordinary three space, a non-singular matrix transformation of the form $Q_j = P_i T_{ij}$ (T a 3 × 3 matrix) is an affine transformation, producing only rotation and scaling. In the special three dimensional space of homogeneous coordinates, such a transformation (where T is now 4 × 4) is called a projective transformation which in addition to rotation and scaling performs translation and a *perspective transformation*, hence its applicability to computer graphics. This transformation is derived as follows: (see Figure 1).

Let an observer be at the origin **O** of a coordinate system and let a display screen S of size 2 scope units by 2 scope units be at position z = cotα, where α is half the angle subtended by the screen. The x coordinate x_s of the intersection with the screen of a ray from a point **P** to the observer—perspective projection from **O** of **P** onto S—is

$$x_s = \left(\frac{hx}{h}\right)\frac{1}{\left(\frac{hz}{h}\right)}\cot\alpha = \frac{hx}{hz}\cot\alpha$$

Similarly,

$$y_s = \frac{hy}{hz}\cot\alpha$$

and let us define

$$z_s = \frac{h}{hz}$$

Then the point represented by

$$\mathbf{P}_s = [hx/hz \cot\alpha, hy/hz \cot\alpha, h/hz, 1]$$

gives as its x and y coordinates the location on the screen at which to draw the perspective projection of the point **P** and as its z coordinate a value monotonically related to distance, suitable for intensity modulation.

$\mathbf{P}_s$ is equivalent to (hz) $\mathbf{P}_s$ = [hx cotα, hy cotα, h, hz]

$$= [hx, hy, hz, h] \begin{bmatrix} \cot\alpha & 0 & 0 & 0 \\ 0 & \cot\alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and we have a matrix which performs a perspective projection on a point represented in homogeneous coordinates.

Translation is achieved by a matrix of the form

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_t & y_t & z_t & 1 \end{bmatrix}$$

and rotation by a matrix of the form

$$\left[\begin{array}{ccc|c} & & & 0 \\ & R & & 0 \\ & & & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

where R is an ordinary three-dimensional rotation matrix.

Typically, several such matrices—rotation, translation, and projection—will be applied in sequence to arrive at a single compound transformation expressed as the matrix product of the separate matrices.

Observe that given any matrix, M, which performs some transformation, a matrix αM, $\alpha \neq 0$, performs the same transformation, since $\alpha(\mathbf{P} \times M) = \mathbf{P} \times \alpha M$ implies $\mathbf{P} \times M$ and $\mathbf{P} \times \alpha M$ are proportional, for any point **P**. In particular, the identity transformation becomes αI, $\alpha \neq 0$.

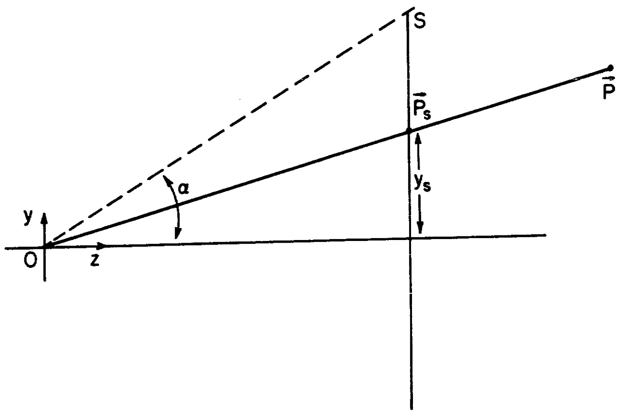


Figure 1—Perspective projection

Surface equation

Let B_{ijk} be a 4 × 4 × 4 array, then the components of a point on a surface are given by a vector function, **P**(u, v), of two parameters, the two degrees of freedom on the surface, as

$$P_k = u^i B_{ijk} v^j \quad ; \quad 0 \leqq u \leqq 1, 0 \leqq v \leqq 1.$$

This expression may be written in matrix notation as

$$P_{hx} = [u^3 \, u^2 \, u \, 1] \, B_{hx} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

$$P_{hy} = [u^3 \, u^2 \, u \, 1] \, B_{hy} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

$$P_{hz} = [\text{etcetera}]$$

$$P_h = [\text{etcetera}]$$

These expressions may also be written without subscripts in the following fashion to indicate all four equations, the implication being that B is a matrix whose components are vectors:

$$\mathbf{P}(u, v) = [u^3 \, u^2 \, u \, 1] \, B \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Endpoint-derivative formulation

Following Coons closely, let F_0, F_1, G_0, G_1 be functions of one variable such that

$$\left.\begin{aligned} F_i(j) &= \delta^i_j \; ; G_i(j) = 0 \\ F'_i(j) &= 0 \; ; G'_i(j) = \delta^i_j \end{aligned}\right\} \quad i, j = 0, 1$$

Then we can define a surface as

$$\mathbf{P}(u, v) = [F_0(u), F_1(u), G_0(u), G_1(u)]\, A \begin{bmatrix} F_0(v) \\ F_1(v) \\ G_0(v) \\ G_1(v) \end{bmatrix}$$

If we insist on F_i, G_i being cubic polynomials, the above conditions define them completely as

$$[F_0(u)\ F_1(u)\ G_0(u)\ G_1(u)] =$$

$$[u^3\ u^2\ u\ 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The 4 × 4 matrix above occurs often and has become identified with the symbol M, for "magic" matrix. The tensor A in the above equation is thus the representation of a surface using the basis functions F_0, F_1, G_0, and G_1.

We can convert between the two forms of representation thus far introduced by means of the identities

$$B = M\, A\, M^T$$

$$A = M^{-1}\, B\, M^{-1T}$$

since $[F_0\ F_1\ G_0\ G_1] = [u^3\ u^2\ u\ 1]\, M$. In these cases the matrix multiplication implied is that obtained by taking each slice of the tensors separately, as $B_{hz} = M\, A_{hz}\, M^T$ and so forth.

(Or, we could write it as

$$B_{ijk} = M_{i\ell}\, A_{\ell mk}\, M_{mj}$$

$$A_{ijk} = (M^{-1})_{i\ell}\, B_{\ell mk}\, (M^{-1})_{mj}\ ) .$$

Because of the definition of the F_i, G_i functions, the components of A relate directly to various partial derivatives of the surface function $\mathbf{P}(u, v)$ as indicated below:

$$A = \begin{bmatrix} 00 & 01 & 00_v & 01_v \\ 10 & 11 & 10_v & 11_v \\ 00_u & 01_u & 00_{uv} & 01_{uv} \\ 10_u & 11_u & 10_{uv} & 11_{uv} \end{bmatrix}$$

where the notation, from Coons, is exemplified by

$$11_{uv} = \left.\frac{\partial \mathbf{P}(u, v)}{\partial u\, \partial v}\right|_{\substack{u=1 \\ v=1}} = \mathbf{A}^{11}_{uv}$$

$$01_u = \left.\frac{\partial \mathbf{P}(u, v)}{\partial u}\right|_{\substack{u=0 \\ v=1}} = \mathbf{A}^{01}_{u}$$

Note well that these derivatives refer to each component of the homogeneous coordinate four-vector; for example,

$$\mathbf{P}_u = \frac{\partial(h(u, v)p(u, v))}{\partial u} = \frac{\partial h}{\partial u}\mathbf{p} + h\frac{\partial \mathbf{p}}{\partial u}$$

and we must solve for

$$\frac{\partial \mathbf{p}}{\partial u} = \frac{\mathbf{P}_u - \mathbf{p}h_u}{h}, \quad \mathbf{p} = [x\ y\ z\ 1], \quad \frac{\partial \mathbf{p}}{\partial u} = [x'\ y'\ z'\ 0]$$

to obtain the partial derivatives of the three dimensional components, x(u, v), y(u, v) and z(u, v).

Curves associated with the surface

By fixing one parameter a curve in the surface is traced by letting the other parameter range over the interval (0, 1).

For example, let v = b, then

$$\mathbf{P}(u, b) = [u^3\ u^2\ u\ 1]\, B \begin{bmatrix} b^3 \\ b^2 \\ b \\ 1 \end{bmatrix} = [u^3\ u^2\ u\ 1]\, A_{ub}$$

where

$$A_{ub} = B \begin{bmatrix} b^3 \\ b^2 \\ b \\ 1 \end{bmatrix}$$

The matrix A_{ub} represents the rational parametric cubic given by the above expression. The properties of these curves are discussed in a companion paper.[8]

The curves $\mathbf{P}(0, v)$, $\mathbf{P}(1, v)$, $\mathbf{P}(u, 0)$, $\mathbf{P}(u, 1)$ are the boundary curves of the surface; in the shorthand notation they are denoted by 0v, 1v, u0, u1. We will denote the matrix describing a curve for u = constant by

$$A_{av} = ([a^3\ a^2\ a\ 1]\ B)^T$$

so that the curve is given by an expression of the standard form,

$$\mathbf{P}(a, v) = [v^3\ v^2\ v\ 1]\ A_{av}\ .$$

For computer display of the surface we generate the family of curves

$$\mathbf{P}_{ub}\ ,\ b = \frac{i}{m}\ ,\ i = 0, 1, \ldots, m$$

or the family

$$\mathbf{P}_{av}\ ,\ a = \frac{j}{n}\ ,\ j = 0, 1, \ldots, n$$

or both. The continuous curve $\mathbf{P}_{ub}$ will be approximated by a series of chords

$$\mathbf{P}_{u_i b}\ \mathbf{P}_{u_{i+1} b}\ ,\ u_i = \frac{i}{k}\ ,\ i = 0, 1, \ldots, k - 1$$

and $\mathbf{P}_{av}$ by the chords

$$\mathbf{P}_{av_j}\ \mathbf{P}_{av_{j+1}}\ ,\ v_j = \frac{j}{\ell}\ ,\ j = 0, 1, \ldots, \ell - 1\ .$$

Iterative display of surface

The successive chords of a curve and the successive curves of a family on the surface are computed using Cohen's finite difference scheme.[8] Without going into details, we compute

$$P_k(m\delta, n\gamma) = \binom{m}{i}\ \delta^i\ C_{ijk}\ \gamma^j\ \binom{n}{j}$$

for an appropriate tensor C dependent on the surface and the particular family of curves to be generated. The multiplication by $\binom{m}{i}\ \delta^i$ is equivalent to multiplication by the matrix

$$\begin{bmatrix} 1 & \delta & 0 & 0 \\ 0 & 1 & \delta & 0 \\ 0 & 0 & 1 & \delta \\ 0 & 0 & 0 & 1 \end{bmatrix}^m$$

which is achieved by simple shifts and additions, and similarly for $\gamma^j\ \binom{n}{j}$. The summations over i and j are commutative. If we are drawing the curves $\mathbf{P}(u, n\gamma)$ we first compute a 4 × 4 matrix $C^n_{ik} = C_{ijk}\ \gamma^j\ \binom{n}{j}$ for each curve and then iterate upon it to compute the successive points $P_k(m\delta, n\gamma) = \binom{m}{i}\ \delta^i\ C^n_{ik}$ on the curve.

Transformations of the surface

In the previous sections the coordinates of a point $\mathbf{P}(u, v)$ on the surface have been defined by expressions of the form $P_k(u, v) = U_i(u)\ S_{ijk}\ V_j(v)$ where $\mathbf{U}$ and $\mathbf{V}$ are vector-valued functions of the parameters. Let T be a projective transformation to be applied to the surface at each point $\mathbf{P}$ and let $\mathbf{P}'$ be the transformed point. Then we have

$$\begin{aligned} P'_\ell = P_k\ T_{k\ell} &= (U_i(u)\ S_{ijk}\ V_j(v))\ T_{k\ell} = \\ &= U_i(u)\ (S_{ijk}\ T_{k\ell})\ V_j(v) = \\ &= U_i(u)\ S'_{ij\ell}\ V_j(v) \end{aligned}$$

where the surface tensor S_{ijk} is replaced by a transformed tensor $S'_{ij\ell} = S_{ijk}\ T_{k\ell}$. This derived tensor, $S'_{ij\ell}$ will generate the transformed surface, without the need to transform each point to be displayed. In other words, the sixteen vectors $\mathbf{S}_{ij} = [S_{ij0}\ S_{ij1}\ S_{ij2}\ S_{ij3}]$ transform in the same fashion as points on the surface and can be regarded as a set of sixteen points which define the surface. Notice, of course, that any transformation—including the identity transformation αI—of the surface must be applied identically to all sixteen $\mathbf{S}_{ij}$. These properties are obviously what allow us the right to call the S_{ijk} array a tensor.

Reparameterization

As with a curve matrix, the tensor describing a surface can be reparameterized to

1. change the rate at which the surface is traversed by the parameters and to
2. display other (smaller or larger) portions of the surface with the same parameter range, $0 \leqq u \leqq 1, 0 \leqq v \leqq 1$.

In the case of surfaces, 1 is particularly important for such a reparameterization changes the appearance of the curvilinear net used to display the surface, by changing the density of contour lines non-uniformly, whereas with curves it affects only the accuracy with which the chords approximate the curve.

For each of the basis functions used so far, $[u^3\ u^2\ u\ 1]$, $[F_0\ F_1\ G_0\ G_1]$ or $[\binom{u}{0}\ \binom{u}{1}\delta\ \binom{u}{2}\delta^2\ \binom{u}{3}\delta^3]$, there exists a reparameterization matrix $S(\alpha, \beta, r)$, a function of three parameters α, β, and r which will

1. map $u = 0$ to $u = \alpha$
2. map $u = 1$ to $u = \beta$
3. change the parameter rate by a factor r.

Two of these reparameterization matrices, $S(\alpha, \beta, r)$ and $S(\gamma, \delta, s)$ can be applied to a surface tensor T to compute a new surface tensor T′ as

$$T'_{ijk} = S_{i\ell}(\alpha, \beta, r)\, T_{\ell mk}\, S_{jm}\,(\gamma, \delta, s)$$

in which the parameterization in both u and v has been altered.

A shape-invariant transformation of the form $S(0, 1, r)$ combined with the identity αI has the property of allowing us to adjust the fourth homogeneous coordinate h at three of the corner points at will; in particular, we could arbitrarily assign $h^{00} = h^{01} = h^{10} = 1$ without any loss of generality in the class of surfaces represented. Correspondingly, if we are given a surface constrained in this fashion we can reparameterize it so as to arrive at a more satisfactory rate of display.

Constructing the surface

The sections above describe some of the properties of a given surface and are thus analytical; the usual question, however, is the constructive problem of generating a surface from external conditions. This and the following two sections sketch some partial solutions that are being investigated.

Suppose we are given the four boundary curves of a surface, specified by matrices A^{0v}, A^{1v}, A^{u0}, A^{u1} in endpoint-derivative form. By the geometrical closure of the curve segments bounding the surface these matrices are guaranteed to *represent* the same corner points **P**(0, 0), **P**(0, 1), **P**(1, 0) and **P**(1, 1) at the intersections of the generated boundary curves. This representation is, however, in homogeneous coordinates and we have no guarantee that the four-dimensional *vectors* given by the matrices are the same. (In other words, $\mathbf{A}^{0v}(v = 0)$ is proportional to but not necessarily equal to $\mathbf{A}^{u0}(u = 0)$.) Using the reparameterization mentioned in the previous section we can however arbitrarily assign absolute values to the fourth homogeneous coordinate h at all four corners, adjusting the curve matrices A^{0v}, A^{1v}, A^{u0}, A^{u1} so that the endpoints will now be represented by identical vectors.

In assigning the four values of h only one degree of freedom will affect the shape of the surface. The other three will be used in specifying the parameterizations in u and v and in assigning an absolute homogeneous scale to the surface.

At this point we have filled in the tensor as shown in Figure 2. (for example, $S_{000} = P^{00}_{hx} = A^{u0}_{00}$; $S_{302} = (P^{01}_{v})_{hy} = A^{0v}_{23}$) The remaining sixteen components can be determined from any sixteen independent conditions; an easy set would be to specify $\mathbf{p}_{uv}$ at each of the four corners and to specify a point $\mathbf{p}_s$ through which the surface must pass at some specified value of the parameters, say $u = 1/2$, $v = 1/2$. With these conditions we can solve for $(P_h)_{uv}$ at the four corners with one degree of freedom left, say to specify the h value reached at the center point $\mathbf{P}_s$ or some other appropriate geometrical constraint.

Instead of specifying $\mathbf{p}_{uv}$ directly, one could specify

$$\left.\frac{\partial \mathbf{p}}{\partial u}\right|_{\substack{u=0\\ v=a}} \qquad \left.\frac{\partial \mathbf{p}}{\partial u}\right|_{\substack{u=1\\ v=b}} \qquad \left.\frac{\partial \mathbf{p}}{\partial v}\right|_{\substack{u=c\\ v=0}} \qquad \left.\frac{\partial \mathbf{p}}{\partial v}\right|_{\substack{u=d\\ v=1}} \qquad \text{(see Figure 3)}$$

as desired outward tangents from the boundary curves, solving for the mixed partial derivatives. In all the above we are talking about conditions expressed in

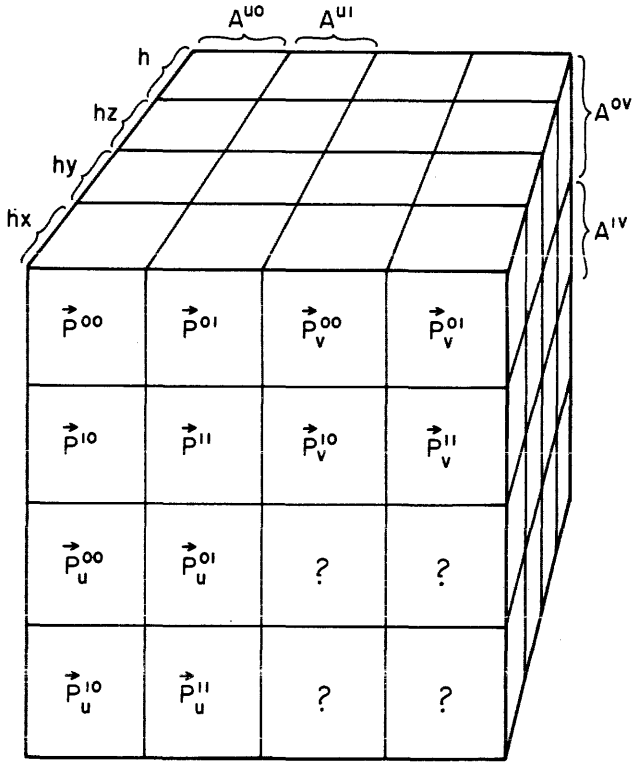


Figure 2—Constructing the surface from boundary curves

terms of three-dimensional points or vectors; the determination of the homogeneous coordinate representation for each such vector is done by specifying other conditions, namely, the point $\mathbf{p}_s$.

Product surfaces and surfaces of revolution

A convenient way to specify certain surfaces is as the "product" of two curves; if the curves are chosen appropriately, a surface of revolution results. Let **P**(u) and **Q**(u) be two curves; then the i'th component of a surface S can be defined as

$$S_i = P_i(u)\, Q_i(v).$$

If we let

$$P_i = \phi_j(u)\, A_{ij}\,,$$

$$Q_i = \phi_k(v)\, B_{ik}$$

where A, B are the appropriate matrices for the basis functions ϕ, then $S_i = \phi_j(u)A_{ij}\, B_{ik}\, \phi_k(v)$ and we can identify

$$T_{ijk} = A_{ij}\, B_{ik}$$

as the tensor representing the surface.

If one of the curves **P**(u) or **Q**(v) is planar, say **P**(u), we prove that all curves of the corresponding family, **S**(u, a), are planar:

Let **A** be the vector representing the plane of **P**(u); define a vector **B**, component by component, as

$$B_i(v) = \frac{A_i}{Q_i(v)}\,.$$

Then

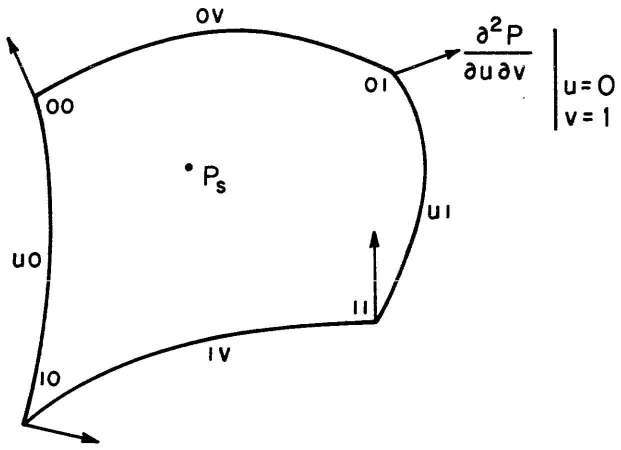
$$S_i(u, v)B_i(v) = P_i(u)Q_i(v)B_i(v) = P_i(u)A_i = 0$$

where now the summation convention holds, and the curve **S**(u, a) lies in the plane **B**(a). Hence, if both curves **P**(u), **Q**(v) are planar, then all the constant parameter contours are plane sections of the surface.

In particular, let **P**(u) be a curve in the x = y plane and let **Q**(v) be a portion of a circle with center [0, 0, 1] in the z = 1 plane. Then we have, in homogeneous coordinates,

$$P_h\, P_x = P_h\, P_y$$

$$(Q_h\, Q_x)^2 + (Q_h\, Q_y)^2 = (Q_h\, r)^2\ ;\ Q_h\, Q_z = Q_h$$



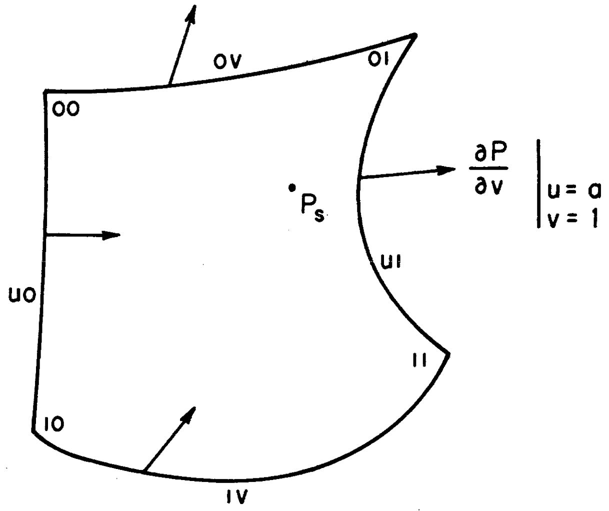


Figure 3—Derivative vectors as conditions on the surface

as conditions on the curves. The surface, in homogeneous coordinates, is

$$h = P_h\, Q_h$$

$$hx = P_h\, P_x\, Q_h\, Q_x$$

$$hy = P_h\, P_y\, Q_h\, Q_y = P_h\, P_x\, Q_h\, Q_y$$

$$hz = P_h\, P_z\, Q_h\, Q_z = P_h\, P_z\, Q_h$$

or, in regular coordinates,

$$x = P_x\, Q_x$$

$$y = P_x\, Q_y$$

$$z = P_z$$

and we have

$$\left(\frac{x}{P_x}\right)^2 + \left(\frac{y}{P_y}\right)^2 = r^2$$

or

$$x^2 + y^2 = (P_x r)^2$$

and also, $z = P_z$, giving a family of circles about the z axis.

All plane sections perpendicular to the z axis will be circles and all plane sections passing through the z axis will have the same shape as the original curve $\mathbf{P}(u)$. Hence, S is part of a surface of revolution constructed by rotating the curve $\mathbf{P}(u)$ about the z axis.

Continuity conditions

In constructing an object from an assemblage of elementary surface patches it is often necessary to enforce certain degrees of continuity at the junction between two patches. In particular, suppose S and T are two surfaces meeting in the common boundary curve B, with the other boundaries as indicated in Figure 4. Suppose we wish the following constraints to be satisfied:

1. The 0v, 1v boundaries have a continuous tangent direction at P and Q. That is, the 0v, 1v boundary curves are continuous and have a continuous tangent line.

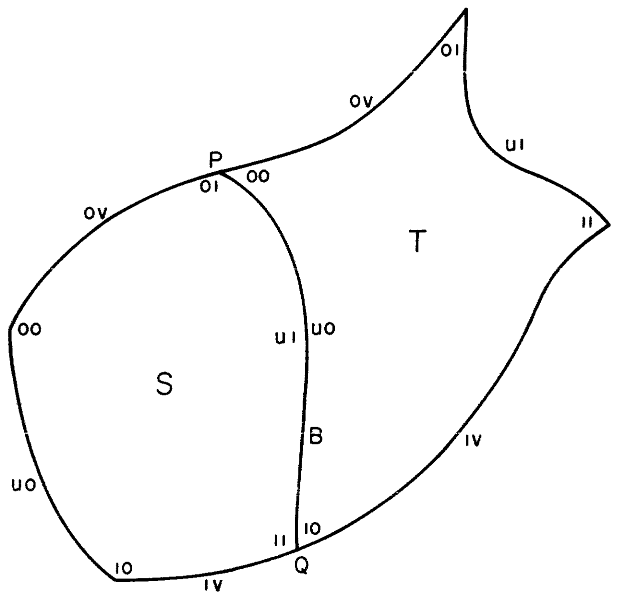


Figure 4—Adjacent surface continuity conditions

2. Everywhere along B the two surfaces have a common tangent plane. That is, the surfaces are continuous and have a continuous unit normal vector.

The first requirements are met by forcing

$$\mathbf{S}_v^{01} \text{ to be proportional to } \mathbf{T}_v^{00}$$

and

$$\mathbf{S}_v^{11} \text{ to be proportional to } \mathbf{T}_v^{10},$$

conditions obtained from the work on rational cubics.

The second constraint is more involved, since it talks about tangent planes. Without deriving it, I state the result that a point $\mathbf{P}$ is on the tangent plane of a surface F at the point $\mathbf{F}(u, v)$ if and only if the determinant

$$|\ \mathrm{P}\ \mathrm{F}(u, v)\ \mathrm{F}_u(u, v)\ \mathrm{F}_v(u, v)| = 0$$

From this, the relation that tangent planes to surfaces S and T coincide can be expressed as

$$|\mathrm{S}\ \mathrm{T}\ \mathrm{T}_u\ \mathrm{T}_v| = |\mathrm{S}_u\ \mathrm{T}\ \mathrm{T}_u\ \mathrm{T}_v| = |\mathrm{S}_v\ \mathrm{T}\ \mathrm{T}_u\ \mathrm{T}_v| = 0$$

where the vectors are evaluated at the appropriate values of the parameters. Since the boundary curve (a rational cubic) is shared we can adjust the parameterization in u such that

$$\mathbf{S}^{u1} = \mathbf{T}^{u0} \text{ and hence that } \mathbf{S}_u^{u1} = \mathbf{T}_u^{u0}$$

The first two determinants will then vanish, so we only consider the third.

In general we cannot expect $\mathbf{S}_v$ to bear any relation to $\mathbf{T}$ or $\mathbf{T}_u$ so the vanishing of the third determinant requires $\mathbf{S}_v$ to be proportional to $\mathbf{T}_v$. But this relation is precisely the same as asking that the function $\mathbf{S}_v(u)$, which represents a curve, be the same as the curve $\mathbf{T}_v(u)$, regarding the derivatives as homogeneous coordinates of points in three dimensions. Since we have already adjusted the parameterizations in u of the two surfaces to be equal along the common boundary, the third determinant will vanish if and only if

$$\mathbf{S}_v(u) = k\mathbf{T}_v(u)$$

for an arbitrary proportionality constant k, and thus the conditions on the tensors are

$$\mathbf{S}_v^{01} = k\mathbf{T}_v^{00}$$

$$S_v^{11} = kT_v^{10}$$

$$S_{uv}^{01} = kT_{uv}^{00}$$

$$S_{uv}^{11} = kT_{uv}^{10}$$

Please notice that, as usual, in all the above we are talking about homogeneous coordinate vectors and that an equation of the form $S_v^{01} = kT_v^{00}$ does *not* constrain the tangent vectors in ordinary three dimensions to be equal. In fact, we have

$$(S_h\,\mathbf{s})_v = (S_h)_v\,\mathbf{s} + S_h\,\mathbf{s}_v = k(T_h\,\mathbf{t})_v = k(T_h)_v\,\mathbf{t} + kT_h\,\mathbf{t}_v$$

where $\mathbf{s} = [x\ y\ z\ 1]$ and similarly for $\mathbf{t}$. Now $\mathbf{s} = \mathbf{t}$ but in general $S_h \neq T_h$, so we have

$$(S_h)_v\,\mathbf{s} + S_h\,\mathbf{s}_v = k(T_h)_v\,\mathbf{s} + k\,T_h\,\mathbf{t}_v$$

or

$$\{(S_h)_v - k\,(T_h)_v\}\,\mathbf{s} = k\,T_h\,\mathbf{t}_v - S_h\,\mathbf{s}_v$$

The fourth coordinate of this vector equation gives

$$(S_h)_v - k\,(T_h)_v = 0$$

and thus

$$0 = k\,T_h\,\mathbf{t}_v - S_h\,\mathbf{s}_v$$

or

$$\mathbf{t}_v = \frac{S_h}{kT_h}\,\mathbf{s}_v\,.$$

Although there is tangent vector direction continuity across the boundary, there is an arbitrary magnitude discontinuity, controlled by the constant k.

Problems for further research

The following paragraphs outline a few of the important questions open for future research. I cover here only the more mathematical questions; clearly the problems associated with organizing these results into an effective, human-engineered system for computer aided design must be answered at some time.

Since a boundary curve can be degenerate, patches with only two or three sides are possible. Such patches by virtue of the degenerate boundary curve must be treated differently as far as tangent plane continuity is concerned. Exactly what constraints can be imposed and what types of surfaces with degenerate curves exist are open questions.

Another question of continuity constraint is the general problem of two overlapping surfaces, say an airplane wing and fuselage. Here we are attempting to ask for surface continuity in the middle of a patch, not on the boundary. Of course, we could use contours in the interior of a patch to define a new subpatch at the boundaries of which continuity is to be imposed. But this may not always be possible, nor may it be the most general solution.

Further results are needed in specifying surfaces from external data. Personally, I prefer the surface-molding approach in which on-line feedback is continuously present as contrasted to the point surface-fitting approach currently used by most surface design systems. However, there are many occasions in which it would be expedient to use such easily displayed surfaces as these to approximate a surface described in some other form, perhaps from measured coordinates.

Even using these bi-cubic rational surfaces as molded surfaces there are not at present sufficient techniques for manipulating the display. For instance, efficient methods of performing small variations in the surface are obviously needed.

I will do no more than mention the question of discovering hidden lines in a figure composed of these surfaces. The problem for this geometry is obviously very difficult and defies a simple analytical solution. Even the application of Warnock's algorithm, effective as it is,[9] may not be the answer, since his techniques have been found very susceptible to Mach band distortion on curved surfaces approximated by planar sections. Such distortion gives the surface a "fluted" appearance at the joints of the planes caused by a psychological enhancement of the intensity discontinuity.

Examples

Figures 5 through 9 have been included to illustrate the appearance of the surfaces. The figures are all true perspective projections of the surfaces from varying vantage points. They should be viewed from such a distance as to give a 45° field of view. They were computed upon a DEC PDP-1 with DEC 340 scopes used for display. The PDP-1 is a rather ancient machine by modern standards, having an 18 bit word length, 5 μsec cycle time, 25 μsec multiply, 40 μsec divide and no floating point.

The time to compute a new view of a surface is from 1 to 2 seconds for these figures, depending on the number of curves used. This is about 1-2 ms. per chord. Each curve is approximated by 32 chords.

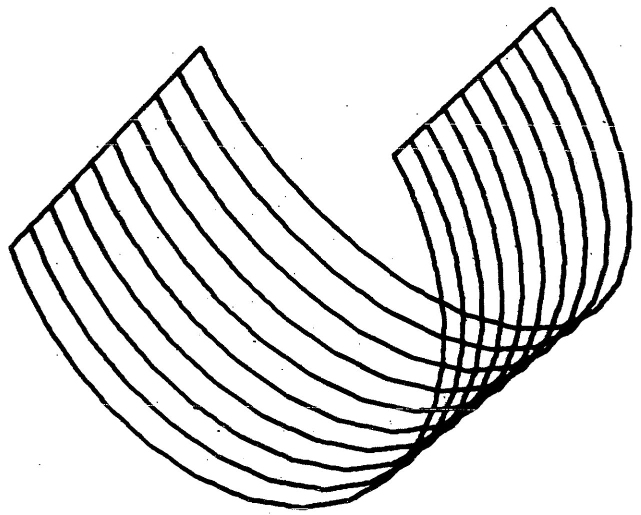


Figure 5—Cylinder

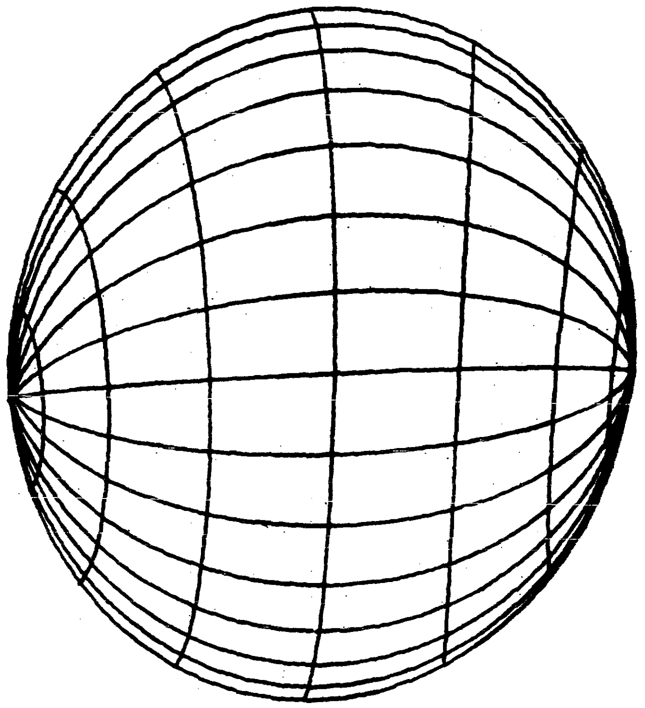


Figure 7—Sphere

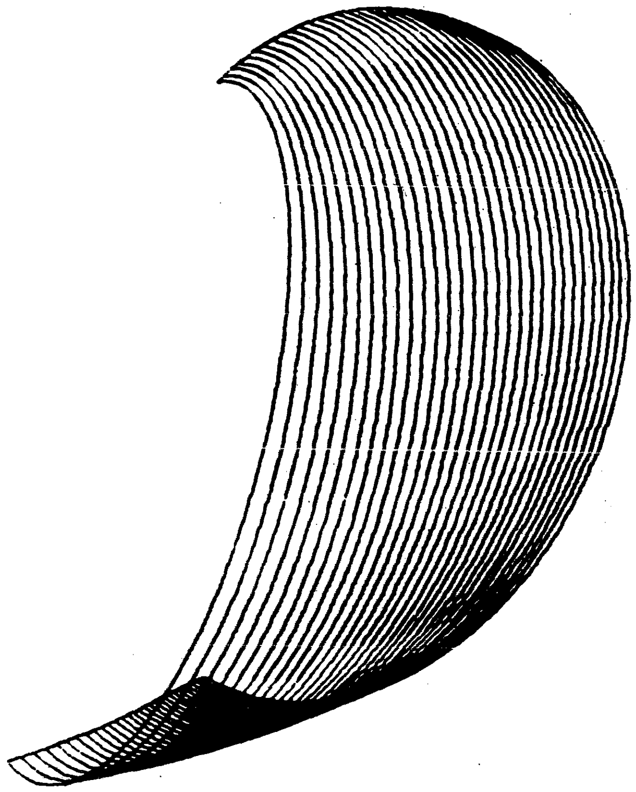


Figure 6—Modified cylinder

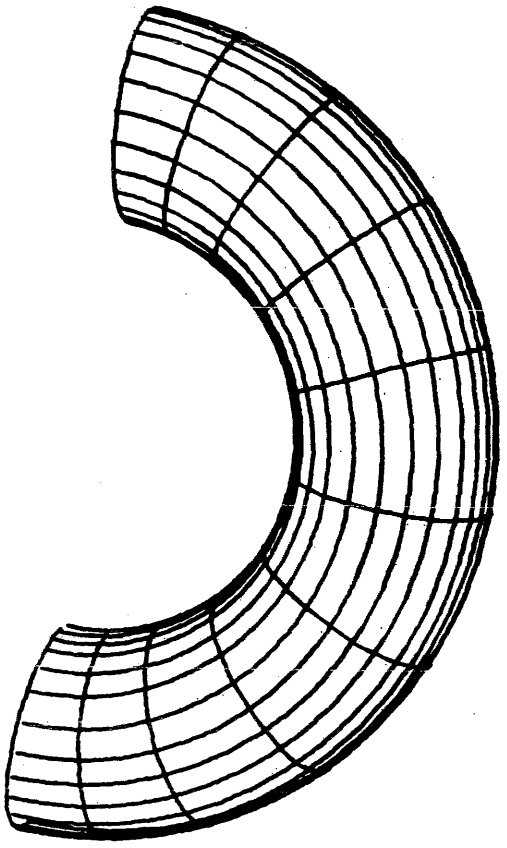


Figure 8—Torus

The tensors for the cylinder, sphere and toroid were manually computed as surfaces of revolution. The hyperbolic paraboloid or saddle surface is simply the equation $z = x^2 - y^2$ expressed parametrically in the

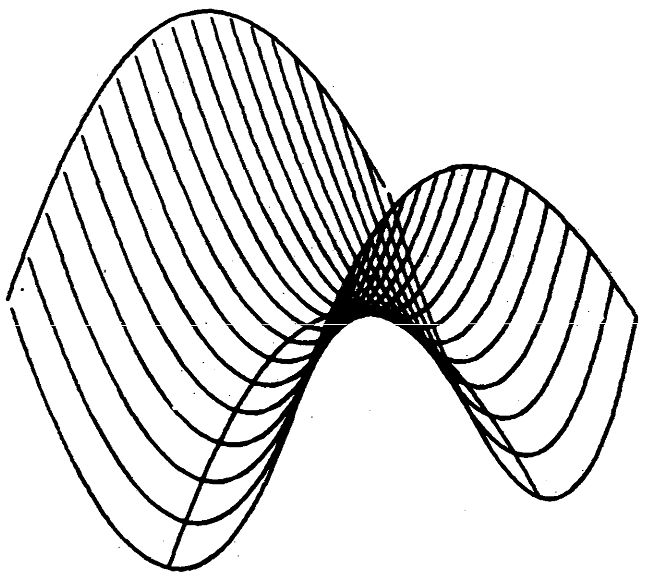


Figure 9—Hyperbolic paraboloid

tensor. The surface shown in Figure 6 was obtained by making a 20 percent random variation in each of the components of the tensor for the cylinder of Figure 5.

REFERENCES

1 R F SPROULL I E SUTHERLAND
A clipping divider
Proc F J C C 1968

2 I E SUTHERLAND
A head-mounted three dimensional display
Proc F J C C 1968

3 S A COONS
Surfaces for computer-aided design of space forms
Project MAC Report MAC–TR–41 MIT June 1967

4 H F BAKER
Principles of geometry
6 Vol Cambridge University Press Cambridge 1922 +

5 L G ROBERTS
Homogeneous matrix representation and manipulation of n-dimensional constructs
The Computer Display Review Adams Associates May 1965

6 E A MAXWELL
General homogeneous coordinates in space of three dimensions
Cambridge University Press Cambridge 1961

7 R M WINGER
An introduction to projective geometry
D C Heath and Co Boston 1923

8 D COHEN T M P LEE
Fast drawing of curves for computer display
Proc S J C C 1969

9 J E WARNOCK
A hidden line algorithm for halftone picture representation
University of Utah Technical Report May 1968 4–5

POGO: Programmer-Oriented Graphics Operation*

by B. W. BOEHM, V. R. LAMB, R. L. MOBLEY, and J. E. RIEBER

The RAND Corporation
Santa Monica, California

INTRODUCTION

Wide-scale application of interactive computer graphics (ICG) is currently inhibited by two major difficulties:

1. Terminal time costs too much.
2. It takes too much effort and expertise to develop and modify ICG programs.

Several approaches, including RAND's Video Graphics System and other television-based or storage-tube based console designs, are currently being tried to overcome the first difficulty.

The POGO system described here represents an attempt to overcome the second difficulty, at least for a certain fairly general class of problems, which includes the interplay of computational programs with alphanumeric and curve input and display, but excludes highly dynamic interplay of computational programs with geometric manipulations.

POGO is fully operational; it allows a user to specify the nature of his ICG interfaces in a natural way at the graphics console itself, simplifying the programming process in two ways:

a. Unburdening the programmer of the tedious and artificial process of specifying ICG control "pages" (CRT displays) by transcribing coordinates from layout paper and stringing together calls to graphic support subroutines.
b. Permitting programmers to create ICG programs without spending a great deal of time learning the intricacies of the graphic subroutine package.

To an extent, these difficulties are removed in more specialized ICG packages—for such applications as simulations, circuit design and layout, trajectory analysis, curve fitting, and chemical analysis—which we would call user-oriented rather than programmer—oriented graphics operations. However, the second difficulty reappears as soon as the user wants some capability not expressible within the standard package (a fairly common occurrence), which can be achieved only by rewriting a piece of the package in a lower-level language.

This paper continues with some background information on the development and usage of POGO, followed by a description of POGO's general capabilities, illustrated by an example of its use.

Background and applications

Background

POGO is implemented on a 256 Kbyte IBM 360/40, furnished with an IBM 2250 graphic display console with light pen, keyboard and function keys, a RAND Tablet[1] for freehand inputs, an SC-4060 hardcopy device, and some IBM 2311 disk drives. It is written in IGS (Integrated Graphics System),[2] a set of FORTRAN-callable routines for elementary graphics manipulations similar to the IBM packages GPAK and GSP.

The "mother" of POGO (in the sense that necessity is the mother of invention) is a user-oriented ICG system for aerospace vehicle trajectory analysis called Graphic ROCKET.[3] This system consists of a network

* This research is supported by the United States Air Force under Project RAND—Contract No. F44620-67-C-0045—monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

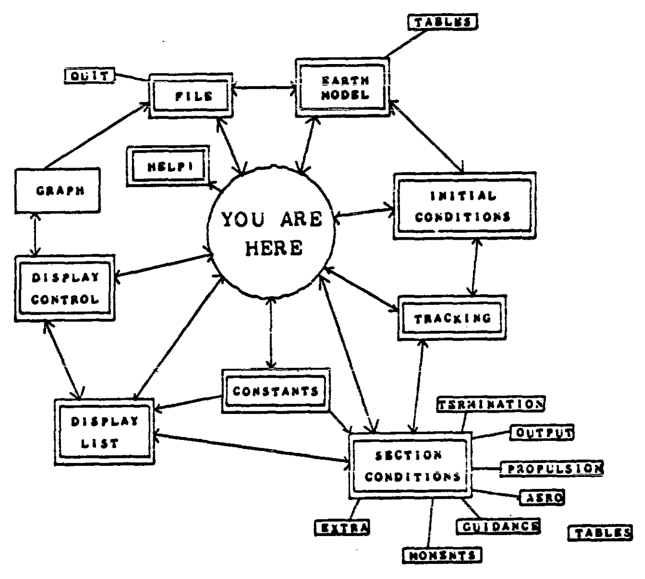


Figure 1—Graphic ROCKET control page

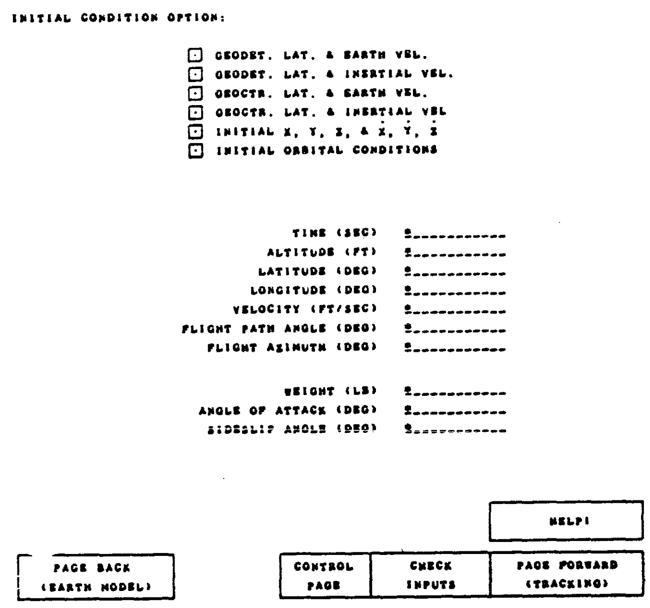

```
INITIAL CONDITION OPTION:

          ⊡ GEODET. LAT. & EARTH VEL.
          ⊡ GEODET. LAT. & INERTIAL VEL.
          ⊡ GEOCTR. LAT. & EARTH VEL.
          ⊡ GEOCTR. LAT. & INERTIAL VEL
          ⊡ INITIAL X, Y, Z, & Ẋ, Ẏ, Ż
          ⊡ INITIAL ORBITAL CONDITIONS


                  TIME (SEC)   $__________
               ALTITUDE (FT)   $__________
              LATITUDE (DEG)   $__________
             LONGITUDE (DEG)   $__________
           VELOCITY (FT/SEC)   $__________
     FLIGHT PATH ANGLE (DEG)   $__________
      FLIGHT AZIMUTH (DEG)     $__________

                 WEIGHT (LB)   $__________
       ANGLE OF ATTACK (DEG)   $__________
       SIDESLIP ANGLE (DEG)    $__________


                                          HELP!

PAGE BACK           CONTROL   CHECK    PAGE FORWARD
(EARTH MODEL)       PAGE      INPUTS   (TRACKING)
```

Figure 2—Graphic ROCKET initial conditions page

of interconnected control "pages" (see Figures 1,2) by which a user specifies the design and flight plan of a rocket vehicle, then specifies and views desired graphical displays of the resulting vehicle performance (Figure 3).

To create control pages for Graphic ROCKET, we had to visualize, usually with the help of a layout sheet, the positions and extent of lines and characters, and to determine their coordinates and character counts to enter as arguments for the appropriate IGS routines. Finally, we had to check the display on the 2250 to see

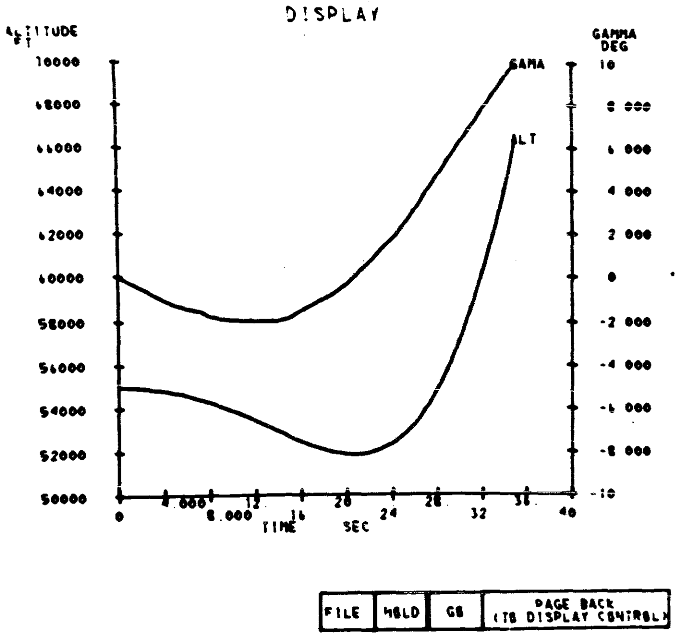


Figure 3—Graphic ROCKET output

if it came out the way we wanted. If not, we had to go through all the steps again. Modifying a control page to incorporate an extra user-desired option also required going through all the steps again.

Finding that other ICG application designers at RAND had similar problems (and having had them previously ourselves),[4] we made a control page design program that was as general as possible without seriously compromising its simplicity. To complete the package, we modified a curve input program that had been developed for a graphic re-entry simulation,[5] and the curve display program and filing routines from Graphic ROCKET, to make them available to the general ICG program developer in forms easy to use, and compatible with the design pages and with FORTRAN computational programs.

Applications

Since then, POGO has been used to create control pages and explanation pages for Graphic ROCKET and other models including on-line simulations and chemical models; it has further been useful for such processes as curve-fitting and digitizing map data. It has also been used to entirely specify the ICG interface for a model of fluid balance in the human body. Using the SC-4060 hardcopy option, which produces a paper copy and a 35 mm film copy of the 2250 scope contents whenever a function key is pushed, POGO has been a handy tool for composing slides for briefings and talks (e.g., Figure 4). In this vein, POGO has an attractive

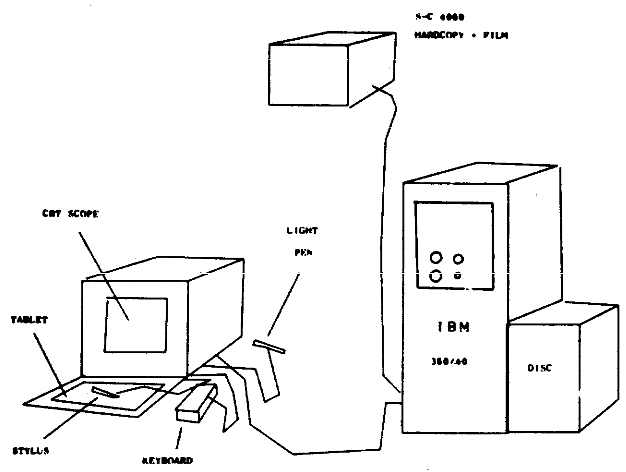


Figure 4—Layout of graphics terminal

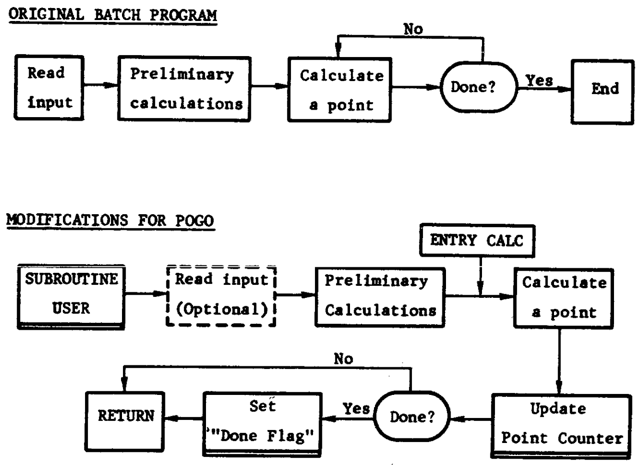


Figure 5a—The POGO process: Batch program modifications

potential for film animation, which we have not yet been able to pursue.

Using POGO: an example

This example involves a model of fluid balance in the human body. The rate of the body's water excretion depends on the level of a hormone called ADH in the blood plasma, while the rate of ADH production depends on the amount of water in the plasma. The functional forms of these relationships are imprecisely known from physiological measurement. One would like to try a set of test functions (curves), specify some external inputs in curve form (e.g., "a drink of water"), and a set of initial conditions, integrate the set of coupled differential equations and compare the results with observed data.

This had been done by means of a batch-type FORTRAN program (Figure 5a) that accepted keypunch input curves and parameters and printed the numerical values of the resulting integration outputs, which were then manually plotted. As this is a tedious and time-consuming process, one would like to give the researcher the capability of tracing his curves directly into the machine and directly viewing the output curves. And, if the fit is not satisfactory, one would like to interrupt the integration, modify the numerical or curve inputs, and try again. This capability is what POGO allows a programmer to create in a simple, natural manner. The accompanying film will show the interactive aspects; this paper will just show some examples of the various pages.

Constructing graphical interfaces

Given any batch program, such as the fluid balance model, producing output functions from input functions and parameters, the following capabilities are needed to construct an interactive-graphics interface:

1. A means of interactively tracing, editing, labeling, and storing input curves.
2. A means of interactively recalling stored input curves and specifying which to use.
3. A means of identifying the types of input curves that may be traced and stored, and of relating them to the appropriate storage arrays in the batch program.
4. A means of interactively selecting, scaling, and displaying output curves.
5. A means of identifying the types of output curves that may be displayed, and of relating them to the appropriate storage arrays in the batch program.
6. A means of entering values of numerical parameters through the graphics console, and of relating them to the appropriate storage locations in the batch program.
7. A means of specifying decision options at the graphics console, including the flow of control between the various input and output pages.

POGO provides these capabilities in two phases of operation at the graphics console, the *design phase* and the *execution phase*, separated by a generally short *programming phase*. The nature of these phases and their interrelations are shown for the fluid balance model example, a fairly typical case, in Figure 5b.

Curve input

Capabilities 1 and 2 for creating, storing, and recalling input curves are provided during the execution

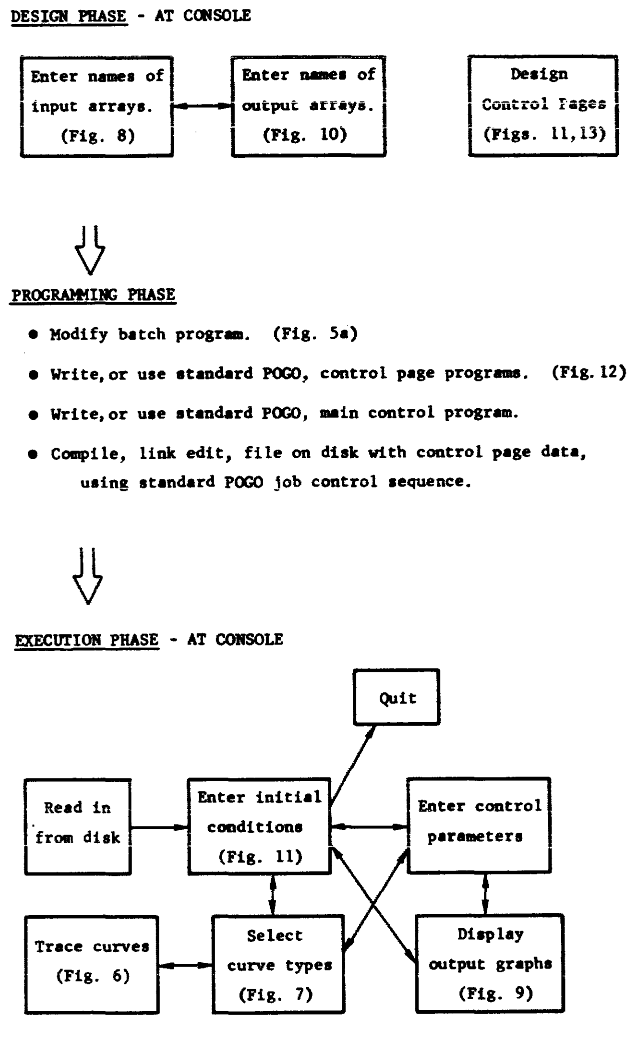

DESIGN PHASE - AT CONSOLE
Enter names of input arrays. (Fig. 8)
Enter names of output arrays. (Fig. 10)
Design Control Pages (Figs. 11,13)
PROGRAMMING PHASE
Modify batch program. (Fig. 5a)
Write, or use standard POGO, control page programs. (Fig. 12)
Write, or use standard POGO, main control program.
Compile, link edit, file on disk with control page data, using standard POGO job control sequence.
EXECUTION PHASE - AT CONSOLE
Quit
Read in from disk
Enter initial conditions (Fig. 11)
Enter control parameters
Trace curves (Fig. 6)
Select curve types (Fig. 7)
Display output graphs (Fig. 9)

Figure 5b—The POGO process: Fluid balance model example

phase by a pair of standard POGO control pages. Figure 6 shows the curve-tracing page. Pushing the "SETUP GRAPH" box allows one to move the axes and adjust the scale numbers to match the curve to be traced in. As the stylus is capacitively coupled to the Tablet, one can place his graph paper on the Tablet and trace the curve by following it on the paper. This capability is provided by the "TRACE CURVE" option. Pushing the "EDIT POINTS" box allows one to specify a fraction of the points to be retained, or to add and delete specified points with the stylus. "ERASE POINTS" removes the curve. "STORE CURVE" writes the curve data on the disk in the location indicated by the legend; this number can be modified, so one may recall a curve, modify it, and keep both copies.

"RETURN TO LIST" transfers control to the other standard POGO input page shown in Figure 7. This page indicates the different types of input curve

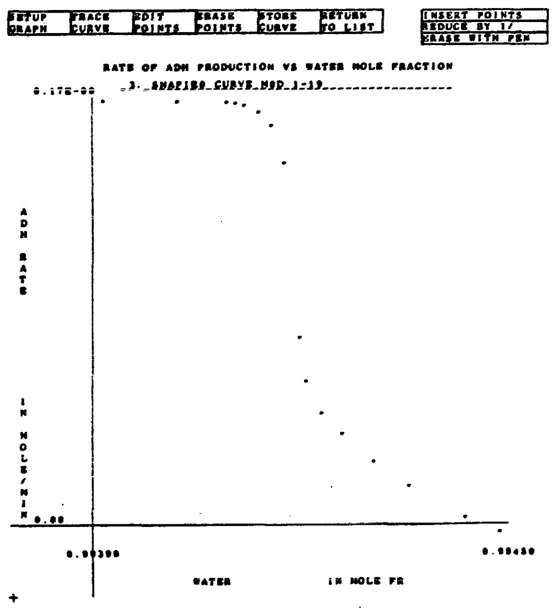


Figure 6—POGO curve-tracing page

available, and the current list of curves of each type appears when the appropriate "LIST" box is hit. One can then specify which of these curves to use in the calculation, to delete, or to display; the latter option transfers control to the curve-tracing page with the specified curve displayed. Up to ten curve types and ten curves of each type can be stored; the "SCROLL" options allow the user to reach the ones not currently displayed.

Capability 3, for identifying types of input curves, is provided during the design phase by the POGO page shown in Figure 8. The user specifies, at the graphics console, all the information POGO needs to label his

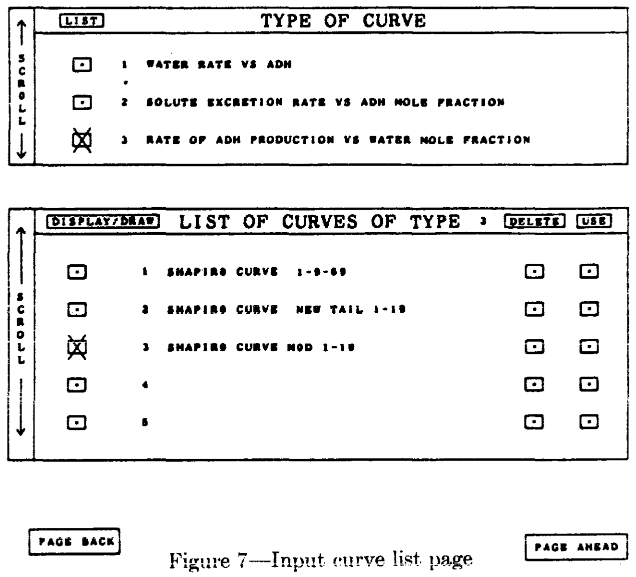
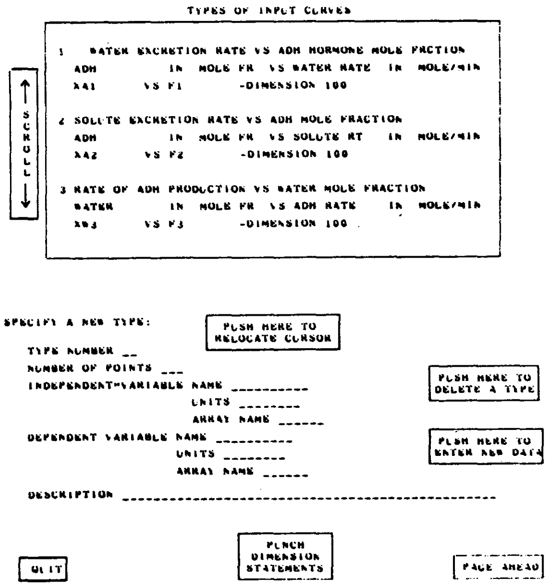


Figure 7—Input curve list page



```
                    TYPES OF INPUT CURVES

   1   WATER EXCRETION RATE VS ADH HORMONE MOLE FRACTION
       ADH          IN  MOLE FR  VS WATER RATE   IN  MOLE/MIN
       XA1      VS F1        -DIMENSION 100

   2 SOLUTE EXCRETION RATE VS ADH MOLE FRACTION
       ADH          IN  MOLE FR  VS SOLUTE RT    IN  MOLE/MIN
       XA2      VS F2        -DIMENSION 100

   3 RATE OF ADH PRODUCTION VS WATER MOLE FRACTION
       WATER        IN  MOLE FR  VS ADH RATE     IN  MOLE/MIN
       XW3      VS F3        -DIMENSION 100

SPECIFY A NEW TYPE:

   TYPE NUMBER __
   NUMBER OF POINTS ___
   INDEPENDENT VARIABLE NAME __________
                        UNITS ________
                        ARRAY NAME ______
   DEPENDENT VARIABLE NAME __________
                      UNITS ________
                      ARRAY NAME ______

   DESCRIPTION __________________________________________
```

SCROLL
PUSH HERE TO RELOCATE CURSOR
PUSH HERE TO DELETE A TYPE
PUSH HERE TO ENTER NEW DATA
QUIT
PUNCH DIMENSION STATEMENTS
PAGE AHEAD

Figure 8—Input array specification page

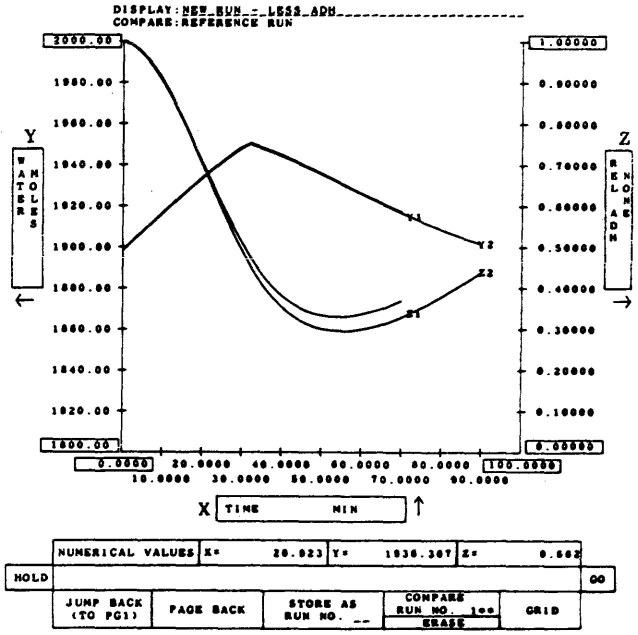


Figure 9—Output graph page

input curves and relate them to the appropriate arrays in his batch FORTRAN program, via the questionnaire form on the lower part of the page. When he pushes the "ENTER NEW DATA" box, the resulting information is summarized on the upper part of the page, and entered into the appropriate master tables on the POGO disk. "PUNCH DIMENSION STATEMENTS" produces a set of punched cards with the resulting FORTRAN array dimensions, if needed, for the user's erstwhile batch program.

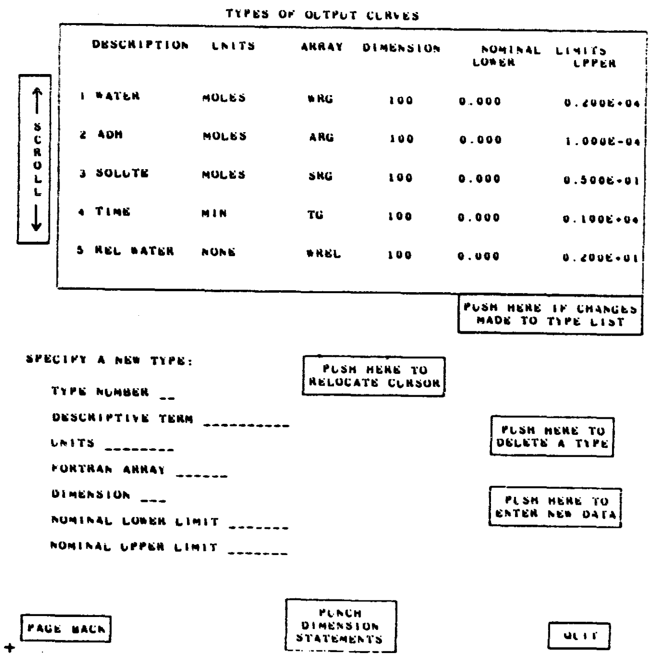
For example, suppose a user has entered the information shown in Figure 8 during the design phase. Then, if, in the execution phase, he hits the "USE" box on line 1 of the "LIST OF CURVES OF TYPE 2" area of the control page of Figure 7, POGO will go to region 2 of its curve input storage space on the disk, read off the values of the independent and dependent variables stored in item 1 of region 2, and store them in the arrays XA2 and F2 (from Figure 8), respectively.

Curve output

Capability 4, for selecting and displaying output curves, is provided during the execution phase by a standard POGO display page shown as Figure 9. This page allows one to graph values of two dependent variables (Y and Z), as functions of one independent variable (X). The windows along each axis show which of the output quantities is being used. Pushing the arrow next to a window brings in the next quantity on the list. Its name and unit appear in the window; its nominal upper and lower limits appear on the axis, and, on pushing the appropriate letter (X, Y, or Z), the corresponding output curve appears.

One may change scale by writing in new values with the stylus, find the numerical values associated with any point in the output region by pushing "NUMERICAL VALUES" and indicating the point with the stylus, or superimpose a grid on the output region. Generally, output points appear as they are being calculated; one may push boxes to "HOLD" the calculation and to "GO" again. And, one can store current curves on the disk and recall them for comparison with future runs by using the "STORE" and "COMPARE" options.

Capability 5, for relating the quantities produced by the erstwhile batch computation program to the routines controlling the Output Graph Page of Figure 9, is handled during the design phase in much the same manner as the corresponding input specification Figure 10. With this page, the user sits at the graphics console and enters, for each variable he wishes to display (up to twenty), its FORTRAN array name and dimension in his computational program, some description for the labels along the axes on the display page, and some nominal lower and upper limits. As with the corresponding input specification page, "ENTER NEW DATA" updates the appropriate master tables on the POGO disk, and "PUNCH DIMENSION STATEMENTS" produces FORTRAN array dimensions on cards, if desired.



TYPES OF OUTPUT CURVES

DESCRIPTION	UNITS	ARRAY	DIMENSION	NOMINAL LIMITS LOWER	NOMINAL LIMITS UPPER
1 WATER	MOLES	WRG	100	0.000	0.200E+04
2 ADH	MOLES	ARG	100	0.000	1.000E-04
3 SOLUTE	MOLES	SRG	100	0.000	0.500E+01
4 TIME	MIN	TG	100	0.000	0.100E+04
5 REL WATER	NONE	WREL	100	0.000	0.200E+01

SPECIFY A NEW TYPE:

TYPE NUMBER __

DESCRIPTIVE TERM __________

UNITS ________

FORTRAN ARRAY ______

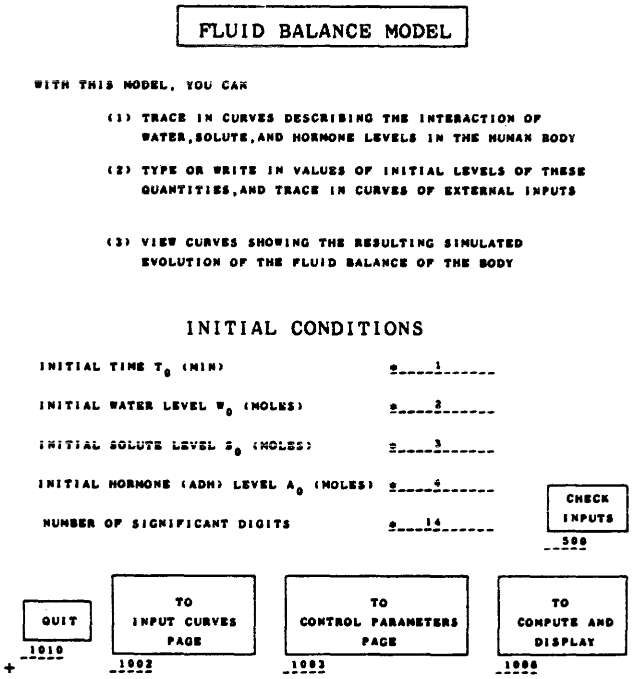
DIMENSION ___

NOMINAL LOWER LIMIT _______

NOMINAL UPPER LIMIT _______

SCROLL
PUSH HERE IF CHANGES MADE TO TYPE LIST
PUSH HERE TO RELOCATE CURSOR
PUSH HERE TO DELETE A TYPE
PUSH HERE TO ENTER NEW DATA
PAGE BACK
PUNCH DIMENSION STATEMENTS
QUIT
+

Figure 10—Output array specification page



FLUID BALANCE MODEL

WITH THIS MODEL, YOU CAN

(1) TRACE IN CURVES DESCRIBING THE INTERACTION OF WATER,SOLUTE,AND HORMONE LEVELS IN THE HUMAN BODY

(2) TYPE OR WRITE IN VALUES OF INITIAL LEVELS OF THESE QUANTITIES,AND TRACE IN CURVES OF EXTERNAL INPUTS

(3) VIEW CURVES SHOWING THE RESULTING SIMULATED EVOLUTION OF THE FLUID BALANCE OF THE BODY

INITIAL CONDITIONS

INITIAL TIME T_0 (MIN)	=____1______
INITIAL WATER LEVEL W_0 (MOLES)	=____2______
INITIAL SOLUTE LEVEL S_0 (MOLES)	=____3______
INITIAL HORMONE (ADH) LEVEL A_0 (MOLES)	=____4______
NUMBER OF SIGNIFICANT DIGITS	=___14______

Figure 11—Initial conditions page for POGO example

Control page design and interfacing

Capabilities 6 and 7, for entering parameter values and specifying decision options at the graphics console, are provided by control pages, which the user designs himself. He can do this at the graphics console by means of a set of POGO routines that allow him to create strings of text, fields for numerical values, option boxes, and geometric figures. He may then use the Tablet stylus to move these around the CRT screen until he is satisfied with the layout. He may enter codes that will be used to relate the numerical fields and option boxes to his FORTRAN program, and then press a button, which has POGO punch out a set of cards that will recreate the display at any later time.

Figure 11 shows one of the control pages that was created with the DESIGN program for the fluid balance model. The number associated with an input field indicates the location in the input storage array into which POGO will store values entered in that field; when one of the boxes is hit, the number accompanying that box will be returned to the user's control program for him to analyze what to do next.

To manage the control pages at execution time, POGO has a standard set of routines that can be incorporated in the programmer's FORTRAN control program in any way he desires. These routines include:

RECALL

given a page number, creates the corresponding page on the screen.

ACTION

waits until the user interrupts via keyboard, function key, light pen, or Tablet, then returns to the control program with numerical codes identifying the type of interrupt and its location.

SAVAL

tests values of variables on the screen to see if they have been changed since the last such test. If so, they are converted to floating point and stored in the location corresponding to their ID number.

By stringing together CALL's to those routines in his FORTRAN control program, the programmer can allow the user to switch from one display page to another, enter new values, select multiple-choice options, or transfer to the curve input and display pages. He generally writes these programs after having composed the displays (see Figure 5), but can work the other way around also.

Figure 12 is an annotated listing of the subroutine

```
      SUBROUTINE CTRLPG(IPG)
      COMMON/PARAM/D(100)
           --storage array for input parameters
      COMMON/JPAGES/JCURVP,JGRAFP,JGRIDP,JUSERP(10)
      COMMON/FLAGS/IPGNXT,NPOINT(20),INPNTS(10),IRUN
           --two standard POGO cards
      COMMON/MODES/Z(200)
           --a communication area for all graphics routines
      DIMENSION BCDVAL(3,15),KVAL(15)
           --for storing BCD images and pointers
      CALL GETIDG(Z,ID)
           --provides local ID number for display
      CALL RECALL(Z,JUSERP(IPG),D,BCDVAL,KVAL,NVAL,DMY,DMY)
           --places display of Fig. 11 on screen
 1    CALL ACTION(Z,ICH,IVAL,ID)
           --waits for user action
      IF(ICH.EQ.2 .OR. IVAL.EQ.500) GO TO 2
           --check inputs on box strike or end key
      IF(IVAL.GE.1001 .AND. IVAL.LE.1010) GO TO 3
           --box strike to change page
      GO TO 1
           --other actions ignored
 2    CALL SAVAL(Z,D,BCDVAL,ID,KVAL,NVAL)
           --convert new numerical entries
      GO TO 1
           --other actions ignored
 3    IPGNXT =IVAL-1000
           --index of next page routine to be called
      CALL DISPLG(Z,0,0,0)
           --clears display screen
      RETURN
           --return to control program
      END
```

Figure 12—POGO control page subroutine

CTRLPG, which uses these POGO routines to manage the initial conditions page of Figure 11 at execution time. First, the page is put onto the screen with RECALL, then ACTION waits for a user action. Suppose he enters the number "43.6" in the field next to "INITIAL WATER LEVEL", and the number "3" in the field next to "NUMBER OF SIGNIFICANT DIGITS" (either via the keyboard or the Tablet stylus), and then pushes the "CHECK INPUTS" box (with the light pen or Tablet stylus). Control will pass from the subroutine ACTION with IVAL = 500; the next statement results in a "GO TO 2" that passes control to SAVAL, which will return with the value 43.6 placed in FORTRAN location D(2) and the value 3.0 placed in location D(14).

The subsequent "GO TO 1" returns control to ACTION, which waits for further action; suppose the user pushes the "TO INPUT CURVES PAGE" box. Control now passes from ACTION with IVAL = 1002; the subsequent tests produce a "GO TO 3," which computes the number of the page to be placed on the screen next (IVAL − 1000 = 2), clears the display screen, and returns control to the main program. Of course, during the programming phase of Figure 5, one must also write a main program that calls the appropriate subroutine when its number is returned.

General comments on control routines

1. If the user's control pages follow the standard format of Figure 12 (number fields referenced to an input array D(100), a "CHECK INPUTS" box with a code of 500, and boxes for going to other control pages, for which page N is given the code 1000+N), then the user need not even concern himself with CALL's to ACTION, SAVAL, etc. In this case he need only insert a

   CALL CTRLPG(IPGNXT)

   in his main routine and include CTRLPG in his load module.
2. Even if the user wants extra features, such as special option or decision boxes, on his control pages, the control routine he writes will be generally simple and straightforward. Furthermore, it will be transparent to such control page modifications as adding or changing commentary, adding new "values" fields, and moving entities around the page.
3. The output display control page of Figure 9 and the input and output array specification pages of Figures 8 and 10 were laid out with the POGO DESIGN program. The corresponding control programs involved CALL's to ACTION, SAVAL, etc., but also required some IGS-level programming for scrolling, curve display management, and numerical values. These figures give some idea of the range of POGO capabilities for composing and managing control pages. This "bootstrapping" capability also allowed us to shorten the development times of the total POGO package, probably by a couple of months; it also makes these pages very easy to modify.

Control page design

All of the figures in this paper, except Figures 3, 5, 6, 12, and 13, were created with the POGO DESIGN program; this gives some idea of the range of its capabilities.

The facilities available to compose and interface displays are indicated in Figure 13, which show the POGO function keyboard layout. To use any POGO facility, the user simply presses the corresponding function key. Short descriptions of some of these facilities follow:

Small Characters

The user indicates with the Tablet stylus where

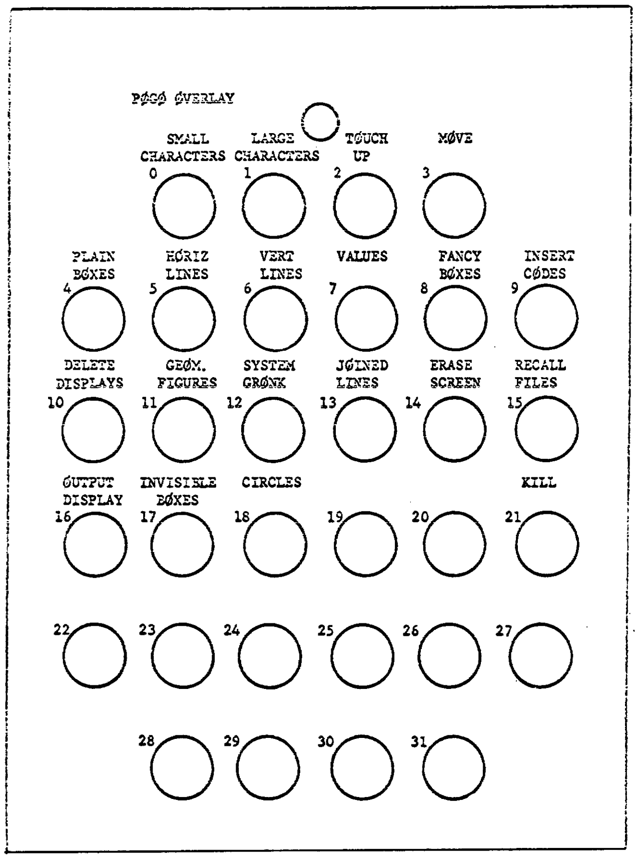


Figure 13—Function key overlay for "DESIGN" program

on the "page" he would like his character string to begin, then enters a string of characters from the keyboard.

Touch-Up

Places the console in the character recognition mode.[6] The user may modify any of the characters on the screen by writing over them freehand with the Tablet stylus.

Move

The user points to the character string or geometric entity he wants to move with the Tablet stylus, and drags it around the screen with the stylus until it is where he wants it. Lifting the stylus completes the action.

Plain Boxes

The user points with the Tablet stylus to define the lower left corner of the box. Pointing again defines the upper right corner; the user may drag the position of this corner around until he indicates (by lifting the stylus) that he is satisfied with the box.

Values

The user points with the stylus to define a place to store the value of a variable; the position is denoted by underscores.

Fancy Boxes

These are similar to plain boxes, except they have a dot at the center to serve as a target for the light pen.

Insert Codes

By each box and each "values" position in the current display, the user is presented with a line of underscores to furnish a numerical code which will identify this box or value this FORTRAN control program.

Joined Lines

The user can draw arbitrary geometric figures consisting of joined line segments with the stylus.

Recall Files

They allow the user to recall a previously created display for review or modification.

Output Display

POGO asks the user to provide a name for the current display. When this is done, POGO punches out a set of cards with the information necessary to recreate the display and identify its components to the FORTRAN control routines.

An example using the DESIGN program: Furniture arrangement

Figure 14 shows the layout of an apartment suite and the outlines of various pieces of furniture. Pushing the MOVE key in the DESIGN program allows one to drag the images of the items of furniture about the screen until he has a satisfactory layout. Or, by using the other DESIGN options, he may create more furniture or modify the outline of the suite. This application makes a nice demonstration of the potential use of interactive graphics in problems of spatial distribution.

The entire application was composed in a twenty-minute console session with the DESIGN program.

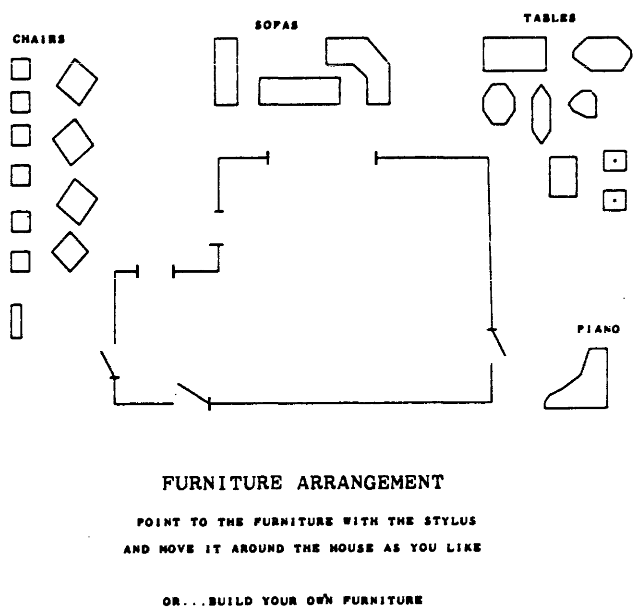

CHAIRS
SOFAS
TABLES
PIANO
FURNITURE ARRANGEMENT
POINT TO THE FURNITURE WITH THE STYLUS
AND MOVE IT AROUND THE HOUSE AS YOU LIKE
OR...BUILD YOUR OWN FURNITURE

Figure 14—A POGO DESIGN application: Furniture arrangement

COMMENTS AND CONCLUSIONS

Versatility

The POGO pages are modular and need not all be used for an application. Thus, a POGO program may consist completely of control pages, as in decision tree applications, or perhaps just curve input and output, as in computing convolution integrals. Further, the modules have clean, well-defined interfaces and can be (and have been) used to design and update the control page parts of a special-purpose ICG applications system, or input and file traced curves for interactive or non-interactive programs.

Although the DESIGN program and the curve-tracing program require the RAND Tablet, the rest of the POGO pages (including pages composed with the DESIGN program)will run on a configuration having only light pen and keyboard input.

Areas for improvement

Some POGO improvements can be accomplished fairly straightforwardly, and these we plan to incorporate, including:

1. Wider range of output pages: charts, histograms, more numerical information.
2. Simple operations on output curves, particularly integrals and derivatives.
3. Reformatting displays at execution time (e.g., for summarizing a sequence of decisions).
4. Scaling and overlay of displays at design or execution time (e.g., for map and network studies).

Others are more difficult, such as incorporating a hierarchical structure in DESIGN program constructs, due to the nature of IGS, our source language. On others, such as a more advanced file and retrieval system, we are waiting for more information on usage patterns.

Finally, one would like to combine the design and execution phases, indicate the flow of control and processing activities at the console along with the page design, and then directly execute the resulting program. RAND is doing some research toward building such a capability, but it is more difficult than POGO by at least an order of magnitude.

Development and usage experience

Total development time for POGO to date has been about one man-year. Machine usage on the 360/40 was about 100 hours for development. As mentioned above, our ability to bootstrap some of the curve input and output pages with the DESIGN program reduced our development time by a couple of months, and makes these pages far easier to modify.

We are just beginning to instrument POGO to measure user interaction and response times. One interesting usage observation is that people tend to get tired of the continual, precise interaction involved in control page design and sign off after one to two hours.

On responsiveness in interactive graphics systems

On our Graphic ROCKET application, we estimate that the POGO DESIGN page has cut our control page development times by factors of four to ten below those required for the manual layout-paper approach. Further, the work is far more palatable, and our error rate is cut to virtually nil.

The most important consequence of the above factors is that they have lowered considerably our responsiveness threshold on providing users with additional capabilities not in the basic Graphic ROCKET package. On most interactive graphics systems we have seen, this extension-threshold is quite high and constitutes a major usage bottleneck.

If there is any general reason for this, we feel it is due to a tendency to design complete ICG systems by deductive inference from an abstract model of typical user performance at a console, producing "closed" systems, which are quite responsive in the small but quite unresponsive in the large. Our experience with Graphic ROCKET and POGO users indicates that general characterizations of user activity are still quite risky, and that more overall responsiveness is gained by the prototype approach: deliberately designing an austere but extendable prototype, then refining it by inductive inference from observed usage patterns.

REFERENCES

1 M R DAVIS T O ELLIS
The RAND tablet: A man-machine graphical communication device
The RAND Corporation RM-4122-ARPA August 1964

2 G D BROWN C H BUSH
The integrated graphics system for the IBM 2250
The RAND Corporation RM-5531-ARPA October 1968

3 B W BOEHM J E RIEBER
Graphical aids to aerospace vehicle mission analysis
The RAND Corporation P-3660 October 1967

4 A S PRIVER B W BOEHM
Curve fitting and editing via interactive graphics
The RAND Corporation P-3742 December 1967
(Also in *Interactive systems for experimental applied mathematics*
M. Klerer and J. Reinfelds Academic Press 1968 343-45)

5 R TURN R L MOBLEY J P HAVERTY M WARSHAW
An application of interactive computer graphics to on-line ballistic missile defense simulation
The RAND Corporation RM-5590-ARPA August 1968

6 G F GRONER
Real-time recongition of handprinted text
The RAND Corporation RM-5016-ARPA October 1966

Computer-aided processing of the news

by J. F. REINTJES and R. S. MARCUS

Massachusetts Institute of Technology
Cambridge, Massachusetts

The process of publishing the news may be divided into three parts: the news gathering stage, the processing of the raw information into publishable form, and the actual printing and distribution of the material. Many technologies, including wire and wireless communication, computers, automatic-typesetting equipments and photographics are being employed by news publishers in order to achieve their goal of getting a hard copy of the news to their readership as quickly as possible at the lowest cost consistent with profitability. Our study of the application of multiaccess computers operated in an online mode to news processing indicates that these machines offer interesting opportunities for departure from the traditional processes employed in the business of news publishing.

In terms of information transfer, newspaper publication embraces four areas: information gathering; information processing; hard-copy reproduction and distribution; and auxiliary business operations. In the news-gathering phase, news and other information are derived from many sources, including staff correspondents and writers, the wire services, advertisement customers, and syndicated columnists. In the news processing phase, information arriving at the newspaper office or generated internally is edited and formatted to reflect the character of each publication and to conform to the depth and breadth of coverage which management desires. In the third, or hard-copy reproduction phase, formatted information is set in type and printed in accordance with the style of the newspaper. In this presentation we are principally concerned with the second area, that is, *news processing.*

As in most manufacturing operations, news processing may be looked upon as a multiple-input, multiple-output operation with a variety of internal feedback loops. Figure 1 illustrates the manner in which news flows from the time of occurrence of a newsworthy event until a permanent record of the event appears as hard copy in the hands of the readership. The process begins with the occurrence of an event or with a decision of a customer to advertise. Advertising itself may be considered as a component of the news in the sense that it is a public pronouncement that someone has an item to exchange in the market place. As indicated in Figure 1, an event may be transformed into a news item by representatives of the wire services, syndicated columnists or by in-house staff correspondents working on a full or part-time basis.

The next phases of the news processing procedure embrace management decisions and production-staff operation which culminate in manuscript and display advertising copy for a specific edition of the newspaper. In many newspapers these materials are now set in type either under computer control or through use of digital techniques. Following type setting, plates are made and hard copy is printed.

Our work to date has pertained to the development of techniques for storing digitally encoded news in a multiaccess computer and for retrieving the stored information through dynamic interaction between the machine and its user. The problem divides itself naturally into four areas. One must first identify the

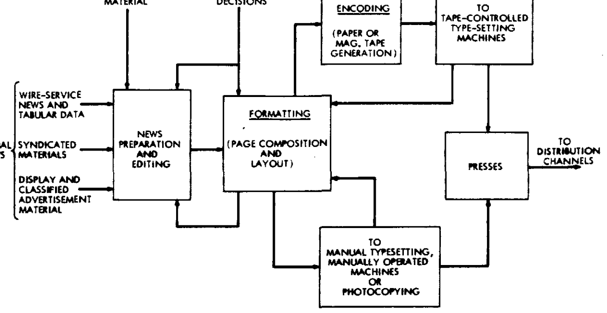

INTERNAL INPUTS
STAFF-CORRESPONDENT MATERIAL
MANAGEMENT DECISIONS
DIGITAL ENCODING
(PAPER OR MAG. TAPE GENERATION)
TO TAPE-CONTROLLED TYPE-SETTING MACHINES
EXTERNAL INPUTS
WIRE-SERVICE NEWS AND TABULAR DATA
SYNDICATED MATERIALS
DISPLAY AND CLASSIFIED ADVERTISEMENT MATERIAL
NEWS PREPARATION AND EDITING
FORMATTING
(PAGE COMPOSITION AND LAYOUT)
PRESSES
TO DISTRIBUTION CHANNELS
TO MANUAL TYPESETTING, MANUALLY OPERATED MACHINES OR PHOTOCOPYING

Figure 1—Block diagram representation of news-processing procedures

parts of a news story which may be helpful to a person who is seeking stored information on a specific topic. Then there is the issue of what procedures to use in order to extract these identifiers from the story. Manual indexing, automatic indexing or a combination of these can be employed. It is also necessary to develop a computer-software system which formats the catalog of information which characterizes the news stories and enables a user to interact with the catalog. Finally, a critical examination must be made of the problems associated with the storing of the full text of news items in the machine.

We have identified 14 items, or *fields*, pertaining to a news story which should be helpful to a person who is seeking to retrieve computer-stored news on a specific topic. These fields are listed in Figure 2 and are divided into two principal categories: Article description fields and subject-content fields. A third category, designated Control Data, refers to items which an indexer may employ to identify the record pertaining to the news item and any information which may be relevant to the indexing procedure. The third category has principal value in an experimental, rather than in an operational environment.

ARTICLE-DESCRIPTION FIELDS

1. Personal News Source (Byline)
2. Personal News Source Title, Affiliation
3. Corporate News Source
4. Headline
5. Edition Statement
6. Newspaper Name
7. Format
8. Length
9. Illustrations
10. Dateline
11. Newspaper Article Location

SUBJECT-CONTENT FIELDS

12. News Category
13. Synopsis
14. Subject Terms

CONTROL-DATA FIELDS

15. Record Number or Identification
16. Input Control
17. Cross Reference

Figure 2—List of data fields for cataloging news articles

The Article-Description fields embrace information about the newspaper issue from which the article was selected, and the location, source, headline, length, and other features of the article. The Subject-Content fields include *in-depth* subject terms, a synopsis of the article and an analysis of the approach or purpose of the article. The Control-Data fields contain an identification number for the news story, inputting data such as method of indexing (automatic or manual) name of indexer, date of indexing and, for analysis purposes, the time consumed in indexing.

Cataloging procedures

Computer-oriented cataloging requires establishment of a flexible structure which can be expanded or otherwise altered as experience is gained with the news-retrieval system. Various items within each field should be coded where appropriate so as to conserve computer storage space, and standard nomenclature, abbreviations and delimiters must be assigned to ensure full recovery of cataloged information. Consider, for example, Field 3, CORPORATE NEWS SOURCE. News sources such as wire services, city bureaus, and other agencies which are responsible for the content of an article are included in this field, and the sources may be identified by one-letter codes. Possible sources and a corresponding code set are:

Source Names	*Name Codes*
Associated Press	a
United Press International	b
New York Times News Service	c
Reuters News Service	d
Washington Post News Service	e
Los Angeles Times News Service	f
Chicago Daily News Service	g
Washington Post—Lost Angeles Times News Service	h
Remote City Bureau (of newspaper being indexed)	r

Thus, in the news catalog code names only are stored within this field; in the retrieval of the cataloged information, the program would decode the symbols into their corresponding source names.

A format must, of course, be designed for each field that contains a multiplicity of information items. Thus, in Field 5, EDITION STATEMENT, two classifications of newspaper editions are identifiable. One is based on time and the other on geographical location. Furthermore, a particular issue must be further delineated through use of information such as volume and issue numbers. Possible edition statements may be derived and given unique codes as follows:

Time Edition Subfield	*Edition Code*
morning final	M
evening final or evening closing stocks final	E
Sunday final	S
weekly	W
one-star	T
two-star	U
three-star	V
four-star	X
five-star	Y
special	Z

Geographical Edition Subfield	
city final	A
suburban final	B
state final	C
New England final	G
New York final	H
East Coast final	I
Midwest final	J
West Coast Final	K
Foreign edition	F

As an example of cataloging Field *5*, consider an item which appears in *volume 35*, *issue 35*, of the *morning statewide* edition of the newspaper. In formatted form this information is entered as follows:

//5/v. 78, no. 35: M/C

A complete set of subfields for each of the fields listed in Figure 2 is given in Reference 1.

Subject-Content Fields. These fields have been designed to give further insights into the subject content of a n ews article and to provide computer-stored "handles"which a user may grasp in order to retrieve desired information.

The NEWS-CATEGORY field (Field 12), indicates the writer's general purpose and may identify such factors as journalistic approach, objectiveness and relevant geographic coverage. Each category with a subfield is designated by a code and might be indexed thus:

Journalistic Approach Subfield:	*Code*	*Category*
		Reportorial
	1	News
	2	Speech
	3	Text (speech, etc.)
		Creative or Interpretative
	4	Editorial
	5	Analysis
	6	Review
	7	Interview
	8	Feature story
	9	Poll
Geographical Coverage Subfield:	a	Local
	b	State
	c	Section of nation
	d	National
	e	International

In accordance with the above classification scheme, an *editorial* on the *Vietnam War* is indexed in formatted form as

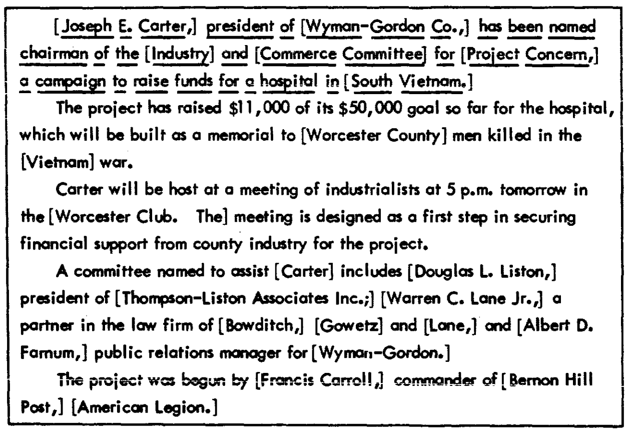
//12/ 4/e

By far the most important component of news indexing is the set of subject terms which is assigned to an article in order to describe its content. We are attempting to make a side-by-side comparison of the relative effectiveness of human indexing and machine indexing. The value of the latter approach obviously depends upon the detail of the algorithms used to extract subject terms. The automatic-indexing program we are using now takes advantage of the nature and style of newspaper writing and anticipates a broad class of words or terms which a user might employ in search of material.

In particular, because the first paragraph of a good newspaper story should contain a summary of the contents of that story, we use as subject terms all the words in the first paragraph except some 13 of the most common function words (*the*, *a*, *and*, *of*, and so forth). Also, much of what a news article is about, and to what a user would presumably wish to refer, is designated by words in the class of proper nouns—names of people, organizations, and places. These nouns are simply captured by extracting all capitalized words. One further class of indexing information is derived from the punctuation and format of a typed news article. Using these clues we can obtain the dateline and byline, where these are present, as well as a substantial set of subject-index terms. In addition, we extract wire-service name and writer's affiliation. Also, we estimate story length on the basis of the number of computer words in text.

This kind of indexing—by extraction of first paragraph, capitalized words, dateline and byline—is evidently quite deep. The ratio of the number of words extracted to the total number of words in the article appears to average about 0.2. Figure 3 illustrates a typical news article and the resulting index terms extracted in accordance with the above rules.

In addition to our using nearly all words of the first paragraph as subject terms, we store the first paragraph, intact, as a separate field, designated Field 13—SYNOPSIS. We suggest that ability to see the first paragraph of a news story will provide a user with a valuable insight to the substance of the story and help him to decide whether or not he wishes to read the entire article.



[Joseph E. Carter,] president of [Wyman-Gordon Co.,] has been named chairman of the [Industry] and [Commerce Committee] for [Project Concern,] a campaign to raise funds for a hospital in [South Vietnam.]

The project has raised $11,000 of its $50,000 goal so far for the hospital, which will be built as a memorial to [Worcester County] men killed in the [Vietnam] war.

Carter will be host at a meeting of industrialists at 5 p.m. tomorrow in the [Worcester Club. The] meeting is designed as a first step in securing financial support from county industry for the project.

A committee named to assist [Carter] includes [Douglas L. Liston,] president of [Thompson-Liston Associates Inc.;] [Warren C. Lane Jr.,] a partner in the law firm of [Bowditch,] [Gowetz] and [Lane,] and [Albert D. Farnum,] public relations manager for [Wyman-Gordon.]

The project was begun by [Francis Carroll,] commander of [Bernon Hill Post,] [American Legion.]

Figure 3—Sample news article showing terms extracted through automatic indexing. Phrases in brackets [ ] are selected by the capitalization algorithm. Underlined words are selected from the first paragraph. Note that all first-paragraph words are selected

Human indexing

In order that a direct comparison of the relative effectiveness of machine and manual indexing can be made, we are indexing the same news stories both ways. The procedures for manual indexing are those which have been established for a companion library-catalog project in our Laboratory, Project Intrex. The basic manual-cataloging procedures are:

- The cataloger is allowed free use of vocabulary and construction in making up subject terms. Generally, subject terms are combinations of noun phrases containing sufficient context to be understood alone.
- Each subject-index phrase is assigned a range number 1, 2 or 3 which designates the extent of the article to which the term applies; range number 1 indicates that the subject term characterizes the entire article, 3 signifies that the subject term applies to a minor part of the article.
- Four additional numbers 4, 5, 6 and 0 may be assigned to special subject terms which may be useful for lookup purposes. Number 4 is assigned to an organization or agency mentioned in the article, but not included in a regular subject term; No. 3 is given to the name of a person *directly* quoted in the article; No. 5 is the name of a person *mentioned* in the article; and No. 0 is a generic subject, broader than specific subject of the article, under which the entire article could be posted. For example, a news story on high-speed trains might be assigned the subject term TRANSPORTATION with a range number (0).

Machine vs. manual indexing

Our experience with 400 news stories indexed manually and 140 articles indexed automatically in accordance with the above guidelines is that machine indexing produces more subject terms per article than does manual indexing; manual indexing yields terms which are longer and which appear to be more interpretive of the article; in manual indexing the range number of a subject-index term is explicit and is one of seven numbers assigned as outlined above; the automatic-indexing procedure assigns either range number 2 for first-paragraph terms or range number 3 for capitalized terms (range number 1 is reserved for headlines, which are not included on our TTS tapes).

The true test of the merit of each indexing method is, of course, its usefulness as a tool for retrieving information from the viewpoints of *relevance* and *completeness*. Measurements of this kind are being planned for our data base and retrieval system.

Software

The software system for storing and retrieving the news data base has been designed to achieve certain requirements which are dictated by user and machine characteristics. From the user's viewpoint the system should be simple to operate and require essentially no time investment in order to make simple queries. Only a small time investment should be required to master its more sophisticated features. The system should also engender user satisfaction, which means that it should be capable of retrieving relevant and complete information expeditiously. Since these requirements are similar to requirements established by Project Intrex for retrieval of library-catalog information, we have adapted the Project Intrex software system[2] to news retrieval. The salient features of our news software system are these:

- Searches are made on single words or word combinations formed by the user in making his request. Users can formulate their requests in their own word style.
- A user needs no prior knowledge of search procedures in order to engage the system. An instruction guide is available online which provides step-by-step information on operating procedures. The guide can, of course, be bypassed by experienced users and instructions for bypassing are available on-line.
- Before information is output the user is informed of the total number of stories found which are relevant to his request and given the option of obtaining the standard output (news-article number, date line and synopsis) or of altering his request. The latter feature is particularly attractive if the amount of relevant material is great and a more specific request is in order, or if an output other than the standard output is desired. In the latter case one or more of the fields listed in Figure 2 may be output.
- As our system is now organized, complete news stories are stored in the machine, minus pictorial and graphical information, and these can be output in full, upon request.
- When a word combination is employed as a request, pertinent news stories are output in order of their relevance to the user's request, that is, if a match is obtained for a document on all words in the user's request, that document is output first. Documents for which matches have been achieved on only some of the words in a request are assigned a lower order in printout.

Formatting and retrieval operations

In order to generate a data base of news articles for our retrieval experiments, we have made arrangements for obtaining TTS paper tapes from personnel at the Worcester Telegram and Gazette, Worcester, Massachusetts. The Worcester Telegram has a Digital Equipment Corporation PDP-8 computer which is used to justify and hyphenate automatically articles prepared on TTS paper tapes. Some of these tapes, and some tapes directly from the wire services which are already justified and which have been selected for incorporation into the newspaper, are sent to us by the Telegram for use in our experiments. Tapes are selected for articles which fall into one or more of the following news categories: The 1968 Presidential Election; The Vietnamese Conflict; The Racial Crisis; and Worcester Urban Renewal. These topics have been chosen because of their interest value during the present time period. Not all articles that go into the paper are being selected because their number would overtax the computer facilities available to the experiments. The category "Worcester Urban Renewal" was chosen as a topic having both local interest and rather general implications. Articles falling within these four categories in the Worcester Sunday Telegram and (evening) Gazette number about 30 to 50 a week. We expect to develop a data base of approximately 1,000 articles over a five or six-month period.

Paper-tape input and code conversion

The paper tapes received from the Worcester Telegram are read into the computer by the Digitronics reader modified to accept teletypesetter (TTS) tapes, which are standard in the newspaper business. (For modification procedure see Reference 2.) The TTS codes are converted to ASCII codes and the full text of the articles is stored on magnetic tape in files of 20 articles each, where this information is maintained for further processing. The articles are then printed out by means of a line printer for inspection and for general hard-copy reference purposes.

In inputting the tapes, the operator also enters the date that the article appeared in the paper. This information is written on the tape at the Worcester Telegram (as is the fact that several tapes make up one article). The operator need enter a given date only once, since the computer program keeps the last date until changed by the operator, and the tapes are batched by date. The computer program automatically assigns each article the next available number as an identifier and stores this information along with the article appearance date, the online inputting date, and the size of the article in computer words as part of a "header block" for each article.

Inverted-file generation

The automatic-indexing programs prepare index terms in a form in which they can be manipulated by standard Project Intrex programs. The first of these operations performs *phrase decomposition*. The subject terms are created originally in multi-word phrases. The first paragraph of a news story is considered an extended phrase. Each string of capitalized words is also considered a phrase. The phrase-decompostiton operation separates these phrases into their individual words and tags these words according to their position within a given phrase, so that the subsequent retrieval operations can suitably account for nearness of pairs of words, if that be desired.

The next operation, *stemming*, deletes endings of words so that in later retrieval operations a user term will still match an index term even though there are minor morphological differences in these terms. For example, a user requesting information about either "bank" or "banking" will be assured of a match if "bank" is in the data base.

A third operation *sorts* these stemmed words alphabetically so that, later, the retrieval operations can take advantage of fast "directed" or "dictionary" searching.

In a fourth operation, the actual *inverted or index files* themselves are generated, together with appropriate directory files for rapid access. In these inverted files are lists of references for given stemmed words. Thus, under the "bank" list are all references to newspaper articles from which the subject words "bank," "banks," "banking," and other forms have been extracted in the indexing process.

A final operation is the *printing of the inverted files* for review and analysis purposes.

Retrieval operations

The retrieval of news articles, or references to them, is accomplished online simultaneously from multiple remote (to the center computer) consoles through use of Project Intrex programs modified for news retrieval. The basic idea is that a user at any one of these consoles types in a subject phrase of one or more words. These words are stemmed and the references to articles from which these stems were derived as index terms are taken from the inverted files.

The computer then reports to the user how many articles matched his subject phrase to a given degree of relevancy. Relevance is estimated by the number of matching words. The user may then request to see the identification numbers of the matching articles or additional information on these articles (for example, dateline, byline, synopsis) or finally, the full text of these articles, which may be stored on magnetic disc or magnetic tape. On the other hand, if the number of matching articles is too large, the user may wish to set additional restrictions; for example, he may specify that his request match only a given byline or only on first-paragraph index terms. This dialog can continue until the user obtains the desired information.

Example of a dialog

A detailed example of an online dialog between user and the computer-stored news system is illustrated Figure 4. The dialog in Figure 4 has been retyped directly from the printout of an IBM 2741 console connected to the M.I.T.-modified 7094 time-sharing computer.[3] For reader convenience the letter *U* has been placed before user statements and the letter *S* before system responses.

Several features of the dialog are worthy of comment. It begins by asking the user to sign in and illustrates the sign-in procedure (1S).After the user has signed in (1U), the system gives the user three options—he can proceed to operate the system, or, if he is unfamiliar with its operation, he can seek aid from a user's guide stored online. We believe a step-by-step assistance plan to help new users is essential during their learning phase. In Figure 4 the user requests information on how to make simple queries (2U). After receiving this information (3S) he proceeds to make a simple request (3U). The system then responds with information that 29 articles have been found relating to his subject (4S). It also indicates to the user what his options are at this point: he may obtain the standard output of ARTICLE NUMBER, DATELINE, and SYNOPSIS for the 29 articles, or change his request completely, or restrict the amount of outputting either by limiting the number of articles outputted or restricting the fields outputted, or finally he may ask for a different type of output information from the standard.

The user decides to obtain the standard output for all articles (4U), and the system responds accordingly (5S). In the illustrated example, outputting was interrupted before the synopsis of the second article was started, and the user requested that information contained in SUBJECT-TERM field for Article No. 295 be outputted (5U). However, after asking for this information and receiving a procedural instruction (6S), the user decided to change his command and to have the information in *all* fields outputted for Article 295 (6U). The information beginning on line 7S is thus obtained. Finally the user asks for the full text of the article (7U) and the system responds accordingly (8S).

Applications to the newspaper business

The concepts of news storage and retrieval presented here carry deep implications for future operations in the newspaper business. That the concepts are directly applicable to newspaper *libraries, archives* or *morgues*, is obvious. A properly organized computer-stored catalog of previously published news, coupled with microfilm storage, or magnetic-disk or tape storage of the

```
1S   Good day! This is News 1a.  Please sign in by typing your name and
     address as in the following example:

     Smith, R. J., M. I. T Room 13-5251

     Note that your sign-in statement should end with a carriage return.

     READY
1U   marcus r s/mit 35 406
2S   If you already know how to use News, you may go ahead and type
     in commands. (Remember, each command ends in a carriage return.)
     Otherwise, for information on how to make simple queries of the
     catalog, type

     info 2

     or, to see the Table of Contents (Part 1) of News-1 Guide which will
     direct you to other parts of the Guide explaining how to make more detailed
     queries, type

     info 1

     READY
2U   info 2
3S   Part 2 of News 1 Guide:  Simple Queries


         To find articles in the system specify your query by subject
     terms as shown in the 3 examples below:
     subject Robert Kennedy
     subject Robert Kennedy, Martin Luther King
     subject Vietcong, Cambodia

         In order to specify additional restrictions (e.g., where author comes from
     or wire service), see Part 9 of the Guide (or type info 9).
     For other than standard output (article numbers, date, and synopsis) or
     for general News command format and abbreviations see Parts
     6.1 and 6.2.

     To see Table of Contents for News 1a Guide and how to use
     the Guide on line, type

     info 1

     Otherwise, you may now make simple queries or use any other command.

     READY
3U   subject nixon
4S   A search on your query SUBJECT nixon found 29 articles. To output the
     catalog fields ARTICLE NUMBER, DATELINE, SYNOPSIS on those articles type

     go

     This output will take about 15 seconds per article. You may terminate this
     output at any time by hitting the ATTN key ONCE. Otherwise, you may change
     your output request. For information see Part 8 of Guide or type

     info 8

     or change your field restriction (see Part 9.5) or make another request of
     News (see Part 1)

     READY
4U   go
5S   3. ARTICLE NUMBER 295

     ( 29)  DATE

     080868

     (46)  DATELINE

     MIAMI BEACH

     (70)  SYNOPSIS

     Richard Milhous Nixon has made his way back to the top of his party. It is
     the greatest comeback since Lazarus, and even in this mean and vicious
     business there is scarcely a Nixon doubter who does not recognize it as a
     remarkable personal achievement. $h


     4. ARTICLE NUMBER 294

     (29)  DATE

     080868

     (46)  DATELINE
     INT. 0

     READY
5U   in number 295/output subjects
6S   To output the catalog fields SUBJECTS on those articles which also match
     your field restrictions INFIELD NUMBER 295 type

     go

     This output will take about 15 seconds per article.
     READY
6U   o all/go
```

```
7S   1. ARTICLE NUMBER 295

     (3)  CATALOGUER

     1

     (4)  ONLINE

     9/11/68

     (21)  WRITER

     By JAMES RESTON

     (27)  NEWSPAPER

     2

     (29)  DATE

     080868

     (34)  LENGTH

     06"

     (46)  DATELINE

     MIAMI BEACH

     (70)  SYNOPSIS

     Richard Milhous Nixon has made his way back to the top of his party. It is
     the greatest comeback since Lazarus, and even in this mean and vicious
     business there is scarcely a Nixon doubter who does not recognize it as a
     remarkable personal achievement. $h

     (73)  SUBJECTS

     Adlai Stevenson (3/01) ;
     Kennedys (3/01) ;
     Nixon (3/01) ;
     PU NEWS ANALYSIS (3/01) ;
     Barry Goldwater (3/01) ;
     Lazarus (3/01) ;
     Many Rejections (3/01) ;
     Harry Truman (3/01) ;
     Dwight Eisenhower (3/01) ;
     American (3/01) ;
     White House (3/01) ;
     Rejected (3/01) ;
     Richard Nixon (3/01) ;
     Richard Milhous Nixon (3/01) ;
     New York (3/01) ;
     America (3/01) ;
     Lyndon Johnson (3/01) ;
     Republican Party (3/01) ;
     California (3/01) ;
     Nelson Rockefeller (3/01) ;

         Output completed. Total of 1 articles found. You may now see
     additional output on these articles by making a new 'output' request (for
     information on how to do this, see Part 8 of the guide or type info 8). You
     may also select a portion of these articles by making a new 'infield'
     request (see Part 9.5).  Otherwise, you may make a new search (see Part
     2) or make other requests (see Part 1).

     READY
7U   o text/go
8S   1. ARTICLE NUMBER 295

     (90)  TEXT


     000295   080868   091168   000015   000015   000321

        3 $h PAGE ONE_JERRY K442 FEW CAN MATCH

           < By JAMES RESTON New York Times News Service

           * MIAMI BEACH _ Richard Milhous Nixon has made his way back to the
     top of his party. It is the greatest comeback since Lazarus, and even in
     this mean and vicious business there is scarcely a Nixon doubter who does
     not recognize it as a remarkable personal achievement. $h

           The politics of America have a way of spinning personal stories no
     rational novelist would dare offer to a skeptical generation. The careers
     of Lyndon Johnson, the Kennedys, Dwight Eisenhower, and Harry Truman, who
     made it to the White House, and of Barry Goldwater, Adlai Stevenson, and
     Nelson Rockefeller and the other also-rans are scarcely conceivable in
     American modern fiction _ and Richard Nixon now joins this unbelievable
     company.

           <Many Rejections

           Rejected by the voters of his native state of California, retired by
     personal choice in an angry farewell from politics in 1962, rejected again
     by the leaders of the Republican Party in his adopted state of New York, he
     has nevertheless won another chance for the presidency, which he lost by
     only 113,000 votes in 1960. PU NEWS ANALYSIS
```

Figure 4—Example of a news-retrieval dialog

full text of the cataloged items should substantially improve the value and usefulness of the newspaper library through quick reaction time and ability to achieve completeness and relevancy of responses to requests. Furthermore, with news libraries available in electrically encoded form, it becomes an easy matter for news libraries to exchange information via wire or wireless transmissions. News banks can be created on a national, regional, state or local level to serve subscriber newspapers. Duplication of archives can be minimized. At the individual newspaper, storage space for clippings can be greatly reduced or eliminated, and the size of the library staff engaged in clipping, indexing, and filing can be correspondingly reduced, particularly if automated-indexing techniques prove successful. Finally, on the basis of continuing reduction in computer system costs, all these advantages will eventually be achieved at a reduction in the overall library budget.

Of importance also is the impact which the concept can make on the operations of the wire services. The present practice of the wire services of supplying to its subscribers an almost continuous stream of news on a "take-it-or-leave-it" basis, can now be re-examined for possible alternate approaches. For example, wire services may choose to transmit to its subscribers only indices of news stories it has generated within the preceding hour (or whatever time block is deemed appropriate) plus a synopsis of the stories that have been prepared either partly or totally automatically. Wire-service subscribers would then access the stories which are computer stored at the nearest wire-service location and request transmission of them in whatever degree of detail they choose to publish. Opportunities for the wire services to perform quick *updating* and *editing* of their stored news should be evident. In contrast to the archival application, in which the data bank is continually growing and therefore demanding increasing computer-storage space and processing time, the wire-services application appears to impose less demand for storage, since files can be purged daily.

Ability to computer index, store and retrieve the news prompts one to re-examine the whole process of writing, editing and formatting the news at the level of the local newspaper office. Here the term "news" is used in a broad sense and includes display and classified-advertisement news. Procedural reorganizations should be aimed toward achievement of higher efficiencies gained through maintenance of the news in digitally encoded form during the writing, editing and formatting stages. Since the trend is toward digitally controlled typesetting equipment, it becomes appropriate to conduct as much as possible of the entire process of news gathering, news processing and news printing in the digital domain.

Storing the news, including advertising information, in digitally encoded form offers interesting new possibilities to newspaper publishers to repackage and resell the news they own. The ease of updating and editing afforded by online interaction with a computer through a graphical console, now makes it attractive to create topical newspapers. The *dedicated newspaper*, devoted entirely to an indepth historical treatment of a single subject of civic, financial, sports or political interest is within easy grasp of each publisher through computer-assisted reorganization of information previously published in his past editions. Once organized on a topical basis, contents of the dedicated newspaper are easily updated by computer means.

ACKNOWLEDGMENT

The research reported here was made possible through a grant to the Electronic Systems Laboratory, Massachusetts Institute of Technology, from the American Newspaper Publishers Association.

REFERENCES

1 S B LAGE R S MARCUS
A cataloging manual for a news-retrieval system
MIT ESL-TM-349 May 1968

2 J F REINTJES R S MARCUS
Computer-aided processing of the news
M I T ESL-SR-348 May 1968

3 M I T COMPUTATION CENTER
The compatible time-sharing system
The M I T Press Cambridge Massachusetts 1965

4 G SALTON
Automatic information organization and retrieval
McGraw-Hill New York 1968

5 M RUBINOFF S BERGMAN W FRANKS
E R RUBINOFF
Experimental evaluation of information retrieval through a typewriter
C A C M Vol 11 No 9 September 1968

6 L HAIBT et al
Retrieving 4,000 references without indexing
Fourth Annual National Colloquim on Information Retrieval May 1967

An on-line information system for management

by G. F. DUFFY and F. P. GARTNER

International Business Machines Corporation
Poughkeepsie, New York

INTRODUCTION

Information for management and specifically Management Information Systems (MIS) have been discussed with increasing frequency over the past few years either by those who have theorized about what an information system for management should consist of or by those who are convinced that MIS from a practical or realistic viewpoint is a myth.

It is the intent of this paper to show that information systems for management are a reality. The devices and software required are now available. It is important that these tools initially be applied on a practical enough level so that extensions and sophistications of MIS are built on a solid foundation. A basic MIS tool consisting of real-time, on-line, remote terminals and a general purpose data manipulation and retrieval software package has been successfully implemented and installed at the Poughkeepsie Laboratory of the IBM Systems Development Division. It is tangible evidence that a viable MIS can be achieved depending on how the tools available are utilized.

This report describes the efforts and problems that preceded MIS installation; the organization and analysis needed for implementing such a system; the basic capabilities of the system; the actual means of making inquiries, updating files, and retrieving necessary data; and a description of system applications.

Prior to time-sharing terminal systems, the phrase "management information" was overused and exaggerated. While several batch-processing-oriented management information systems were of some value, they did not fulfill management's basic information need—*to satisfy the changing and unique needs of management at that moment in time when the information is required.*

The design for a batch processing system must be limited to some realistic level since any sizable data base can be manipulated in thousands of different ways. The systems designer, in the case of batch processing, evaluates the needs of the user. He provides output and a means of input that can be satisfied through normal data processing and, more importantly, can be readily assimilated by the user once he receives the output. Since the output satisfies only a fraction of the manager's changing needs, he exercises one of three options: 1. He provides clerical people to massage normal output into the desirable form. 2. He uses more intuition and "guesstimating" than he would like. 3. He requests program changes that will satisfy his need but may take weeks to implement. The third option is often not followed because the manager realizes that by the time the system is modified, he has an entirely new set of needs which were not and probably could not be anticipated.

The significant point is that batch-oriented data processing is not dynamic enough to provide true management information. Systems analysts erred in assuming that information for management was simply the next logical extension of batch-type data processing, and efforts to take this approach resulted in frustration and the notion that management information systems are a myth. The primary goal, therefore, was not to reshape batch processing but to develop and apply new tools. These tools are now available in the form of remote-terminal systems which utilize generalized sets of data manipulation and retrieval programs. These systems give us the capability to effectively respond to the everchanging information needs of management.

Another important point with respect to the design of a management information system was the assumption that such systems should be oriented to top management, with some fallout to lower management levels.

Contrary to this position was the belief that satisfying the first and second levels of management is most important, initially, with an extension to top management at a later date. In most application areas, satisfying the requirements of operating management indirectly affects the type, content, and timing of information received by top management.

The MIS/360 System installed in the Poughkeepsie Laboratory handles various applications, including engineering, programming, and administration. The computer is time-shared, having both typewriter and display terminals on-line. Seventeen application areas utilizing 38 terminals can interrogate their respective files during prime shift. Inquiry can be made against any record and any field in each file even though that type of inquiry may never have been made before. Certain inquiry format rules must be followed, but the inquiry language is defined by the user and stored in the system. Full English-language terms or abbreviations can be used according to the desire of the MIS user. No knowledge of programming is required.

Efforts by the Systems and Procedures department (S and P) to implement a management information system in the Poughkeepsie Laboratory began in early 1966. Various terminal systems were evaluated and the potential for MIS at this facility was assessed. The system decided upon was under development by a systems programming group at IBM's Corporate Headquarters and intended for use with IBM System/360. The total effort combined the efforts of Corporate Headquarters and several other IBM facilities (including S and P programmers from the Poughkeepsie Laboratory). During 1966, this laboratory started preliminary application analyses for purposes of system justification and obtaining approvals for the ordering of necessary MIS/360 equipment. In 1967, the effort concentrated on preparing for installation in early 1968. At the same time presentations, demonstrations, and in-depth systems analysis were conducted in those application areas that could most effectively use the MIS capability. The MIS system became operational in April of 1968.

One important aspect of MIS implementation was that in the Laboratory a base for planning and justification had been provided through the operation of the Administrative Terminal System (ATS). Basically, this is a text-processing system that allows a terminal operator to enter, retrieve, correct, and store text or data, and also permits automatic formatting with output from a remote terminal or from a high-speed printer. Many of the problems faced in the installation of MIS were similar to those encountered during the ATS installation period.

Basic approach and ground rules for MIS implementation

The successful implementation of MIS can be attributed to the basic approach that was taken. To ensure a solid base emphasis was placed on developing the most productive applications first. Only those applications that provided some analysis support of their own were incorporated, and the initial utilization of MIS capabilities by each area was to be both simple and practical. Each application analysis had to fit within the manpower available and within a time frame that would allow full relief of MIS expenditures at year-end. Certain key factors were involved:

- Organization that provided a focal point for MIS system analysis, programming support, and overall control.
- Personnel, with the knowledge, experience, and motivation oriented toward MIS terminal system concepts.
- Working relationships between MIS systems people and those involved in related batch processing efforts that converged to a single point of responsibility and authority.
- Close agreement on objectives existed between the MIS systems group and the Computer Center management. This permitted both rational MIS cost allocation and proper utilization of the MIS computer.
- Provisions were made for resolving communications problems among MIS users, the Computation Center, customer engineering, the local telephone company, batch processing programming groups, the Accounting department, and the MIS terminal systems group.
- A set of ground rules defining a basic MIS position were established prior to MIS implementation. Potential users clearly understood the MIS capabilities available and were able to make a decision relative to commitment.

MIS ground rules for MIS implementation

The following MIS position was established at the start of the in-depth systems analysis of potential MIS applications:

1. MIS was to be used primarily as an inquiry and retrieval tool with on-line updating as restricted as possible.
2. Only basic MIS generalized programs were to be used, with no user-exit routines or unique-to-Poughkeepsie routines allowed.
3. Total system operation was to be optimized in terms of performance and in the criteria used to develop each MIS application.

a. Minimized response time through the most efficient file organization and core allocation.
b. Reduction of total MIS costs to provide the lowest possible cost to the user. This was partially accomplished by using the same computer for background processing.
c. Concentration on applications having the greatest potential return for the lowest investment, paying close attention to possible MIS networks.
d. Selection of application areas most responsive to MIS and capable of providing some of their own systems support during the analysis and installation period.

Within a system such as MIS, employing generalized programs, there is considerable pressure to provide additional features that satisfy unique user needs. Because of the various interrelationships that exist in MIS-type systems and the many questions about performance of a new system, taking a position other than that described under "Ground Rules" could be disastrous. The resulting confusion could cause a loss of confidence in MIS that would jeopardize the entire systems effort.

Basic MIS relationships

Response time *vs* number of active terminals *vs* transaction load on the system *vs* systems cost are the basic relationships that must be considered in an MIS system. These relationships are affected by several factors.

- The amount of background processing that can be allowed without degrading MIS performance.
- The size of files in the system and the overall storage requirements.
- The number of MIS routines that should be resident in core.
- The complexity and volume of inquiries expected from users.
- The files that are interrogated the most and the least.
- The distribution of activity during on-line time.
- The equipment changes that will decrease access time or increase processing overlap, thereby resulting in faster response.
- The question of whether fields within a file should be indexed or not indexed.
- The extent to which records in a file should be blocked.
- The most efficient file organization.

Network concept

The plan was to concentrate on applications that would develop into a network of terminals. It was believed that these applications had the greatest potential and would make the most impact on the laboratory from a cost, manpower, and communications point of view. Intangible benefits would result from "a better way of doing business" for this laboratory, as well as other IBM facilities closely related to a centralized responsibility at Poughkeepsie.

Integration of MIS functions with batch processing

The approach taken considered such objectives as using MIS to either replace existing batch jobs, or, in the case of new systems design, first utilizing the capabilities of MIS as much as possible and then satisfying the remaining application needs by programming for batch processing. Most of the applications in MIS satisfy these objectives in varying degrees. As MIS capabilities grow, the batch processing required should steadily decrease.

Equipment and configuration

MIS/360 is a real-time, time-sharing system that services both typewriter and display terminals at remote locations. The system configuration (see Figure 1) includes a multiplexor channel with the Transmission Control Unit (TCU) as the prime terminal interface, plus a 512K Central Processing Unit (CPU), 466 million bytes of direct access storage, and the normal Input/Output (I/O) devices.

MIS systems programs

Generalized software for MIS consists of control programs, application programs, and support programs, which are basically tabling and load programs. The control and application programs are fundamental to the system and are described in the following paragraphs. To show how the control and application programs and their associated control areas reside in core, a core map is shown in Figure 2. This map assumes that all routines are resident.

Control programs

Operating system

The system control programs are the latest release of the IBM System/360 Operating System using the MVT option (Multiprogramming with Variable Tasks).

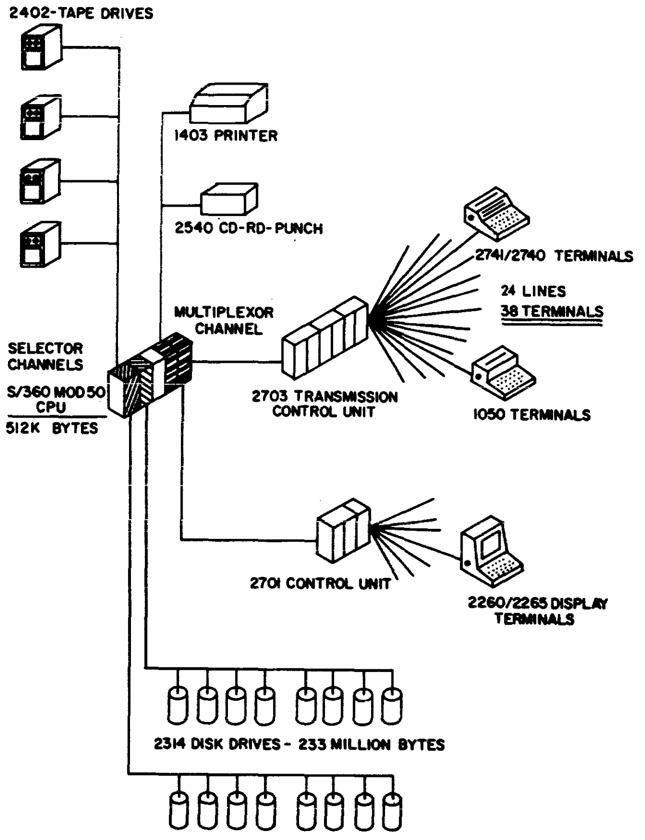


Figure 1—System configuration

The use of MVT maximizes MIS performance and also allows background processing when necessary.

Message processor

The I/O control is handled by the Message Processor module (MP). This module contains all device-dependent routines which support the typewriter terminals for the IBM 2741 Communication Terminal and the IBM 1050 Data Communication System, and the display terminals for the IBM 2260 Display Station. These routines receive each inquiry and, through a data-queuing technique, pass each inquiry to other routines in the MP which, in turn, acts as a buffer between the terminals and the application programs.

Application programs

While the term "application" refers to these programs, actually they represent a set of generalized programs that service all MIS user areas on the system in the same manner. Each user's file or data set consists of a different number of records. There are unique record formats for each file, but the inquiries from each user into his unique file are all handled by the same general-

OS/360 MVT	Message Processor-MP (I/O Control)	MIS Modules	Background Processing
Nucleus	MP Routines	Scheduler*	Peripheral or Batch
		Inquiry Analyzer*	
Link-Pack Area	Device-Dependent Routines (1050, 2740/41, 2260)	Define	
Master Scheduler		Security Analyzer	
	Paging Buffers	Index Search	
Queued Buffer Area	Terminal Control Blocks	File Search and Report Generator	
	Access Methods	Compute	
		Update	
		File Control Blocks	
150K	70-90K	110-240K	
	See Note 1	See Note 2	

Note 1 - Amount of core required for either MP or MIS routines varies according to the number of files and terminal lines on the system, and whether certain programs operate in the overlay mode.

Note 2 - Only MIS routines with an asterisk must be resident in core.

Figure 2—Core storage allocation

ized programs. Most of the MIS routines can operate in the overlay mode when necessary to conserve core space. (See Note 1 under Figure 2.) A brief description of each of these application programs follows.

MIS supervisor—MIS/010

Controls processing of all inquiries, provides time-outs and alternating processing of inquiries, and contains communications areas for system parameters. Each of the other application routines communicates with this module for I/O operations.

Inquiry analyzer—MIS/040

Checks the terms in each inquiry statement against the file description tables, stores the system parameters generated, makes a preliminary security check, and decides when the supervisor should call other required routines such as index search.

Security analyzer—MIS/050

Checks the security code of the user when special access to certain fields within a record is indicated. A violation produces an error message from the error analyzer, creates a history record, and terminates the inquiry.

Index search—MIS/060

Called by the inquiry analyzer if the inquiry includes an indexed field. The index search routine searches the MIS indices and outputs a string of addresses for all records that meet the selection criteria.

File search/report generator—MIS/070

Provides either a full file search of all records in a file, or it searches only those records having addresses passed to it as a result of the index search. Each record is processed against the inquiry selection criteria. If there is an equal compare, the output data are passed to the MIS supervisor program.

Compute/logic and compute/arithmetic routines are also included in this module to handle all CALCULATE requests included in an inquiry statement. These routines are used to develop and store answers in the MIS internal control blocks.

Define—MIS/030

Adds, deletes, or prints user definitions generated at a terminal whenever the inquiry statement includes the DEFINE, CANCEL, or PRT/DEFINITIONS key commands.

Error, inquiry history, and mail processor

Generates either error messages or appropriate end-of-job messages and, at the same time, writes out the inquiry history record. This module also searches the history file for user mail requests when the computer operator is ready to satisfy these particular requests

Update

Changes the contents of fields in a record when activated by terminal inquiries. Editing of data being changed includes alphanumeric and range checking, and an audit trail of all transactions is also supplied.

MIS control tables

Each inquiry statement is compared against the following tables:

Report description
User description
Field description
Access description

These tables describe all pertinent characteristics of each standard MIS report, each field in a record, each user who will access particular files, and the description of all users who have special access to files.

System operation

Each user's file is located on an IBM 2314 Direct Access Storage Facility as are all description tables and the indices for all files. All data sets, including MIS programs not in core, and files, file tables, and indices can be located on assorted storage devices depending on contention problems and channel utilization. The system configuration of the MIS/360, Model 50, used in the IBM Poughkeepsie Laboratory is shown in Figure 1.

Getting on-line

Each user with a dial-up data phone dials the number assigned to him and establishes a link with the computer via the 2703 TCU. If the line is dedicated, rather than dial-up, the user merely puts power-up on his terminal, and he is ready to transmit. In the first case, there is no contention possible on a user's line until he disconnects, and with a dedicated line no contention is possible.

Basic inquiry procedure

The basic elements of any inquiry, shown in Figure 3, are:

Security code
File name
Inquiry parameters
  Format terms
  Selection criteria
  Control terms

If, for example, there are 250 fields in each file record, any of the field terms can be included in the inquiry in any order, as can any selection criteria and control terms desired. The format of the output is dependent on the order of terms within the inquiry.

Definition of inquiry elements

Security

The first entry for any inquiry statement must be the security code assigned to each MIS user. Periodically, the security codes are automatically changed using a random means of selection.

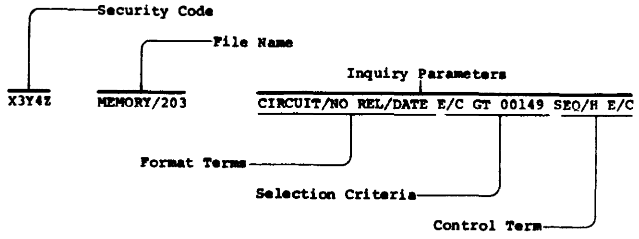


Figure 3—Basic inquiry elements

Special access within a security code

Each security code can be given "special access coding." In the file table, cards containing security codes that indicate special access to certain files or data fields can be specified. For example, many users can interrogate the personnel file, but only cleared users such as salary administration are able to retrieve salary-field data from this file.

Format terms

Each data field in the file has a specific name assigned by the user. Only fields (such as CIRCUIT NUMBER, RELEASE DATE and E/C NO., as shown in Figure 3) included in the inquiry are displayed in the output for those records meeting the selection criteria.

File name

Each MIS user assigns a name to his file and, if desired, the names of any standard reports he wants. These reports must be defined within the MIS tables and, at the time of inquiry, may be retrieved by entering the file or report name. If inquiry parameters are added for fields not in the first standard report, the output will not be a standard report but will include only those terms indicated in the inquiry.

There are three names assigned to each field in the file: normal field name, the term used when making an inquiry, and the column heading for each field when displayed. Each output report includes a heading for each field.

Selection operators

Select terms are:

EQ	— Equal To
NE	— Not Equal To
GT	— Greater Than
NG	— Not Greater Than
LT	— Less Than
NL	— Not Less Than
WL	— Within Limits
OL	— Outside Limits

Multiple selection criteria are allowed on various fields. For each criterion, except WL and OL, the select term must be preceded by a format term and followed by a literal (see Figure 3). Two literals must follow WL and OL.

Control terms

After the desired terms and selection criteria are determined, the form of the output may be further qualified by the use of the following control terms.

TOP	— Top number of records that meet criteria.
LOW	— Lowest number of records that meet criteria.
SEQ/H	— Sequence, high-to-low, of all records meeting criteria; field to be sequenced must be indicated
SEQ/L	— Sequence, low-to-high.
TOTAL	— Provide field totals only, without printing details.
COUNT	— Provide only a count of the number of records that meet criteria.
LIMIT	— Control the number of output lines, i.e., print only a specified number of lines regardless of the number of total records that meet the criteria.
MAIL	— If this term appears in the inquiry, it cancels output from the terminal, and a mail request is stored on the history tape. After MIS is off-line, the operator searches the history file for all mail requests, prints them out on the IBM 1403 Printer, and then mails the output to the user. This is used primarily for an excessively large output and frees the terminal for further on-line user inquiries.
CALC	— MIS terminal users can also perform calculations on data fields and print and total the calculated data that are requested in the selection criteria. Calculate operations include: + (addition), — (subtraction), * (multiplication), / (division), V (square root), ** (exponentiation)
DEFINE	— This is one of the most powerful control terms. It enables the user to equate a report name with a group of terms or with a total inquiry statement. For example, a complex

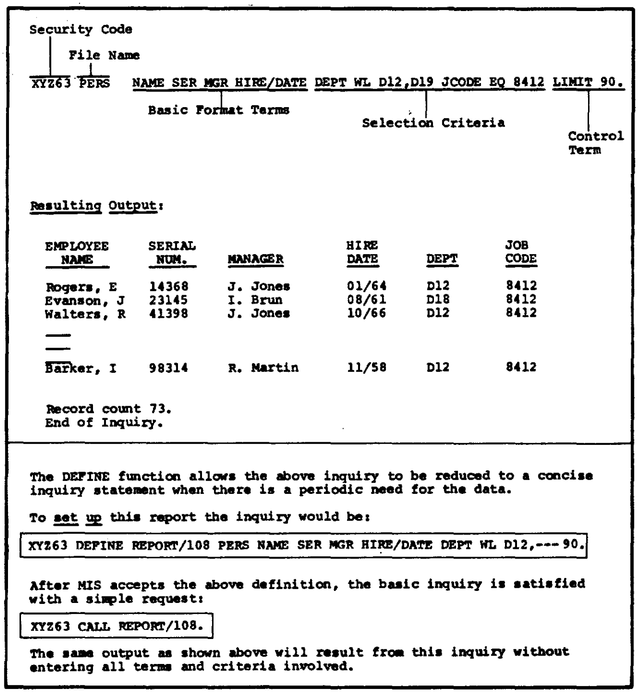
inquiry statement of the maximum length of 132 characters could be equated to a report name such as REPORT/101. The DEFINE capability is used most effectively when a report is frequently required, and the user does not wish to enter the lengthy statement each time. This also eliminates possible operator errors when entering a long inquiry.

Figure 4 shows a typical inquiry into a personnel file, including the report heading and the detailed output. The actual Poughkeepsie Laboratory personnel file contains approximately 4000 records with 248 fields per record. All of the fields can be selectively manipulated with the resulting data output appearing in any desired order or quantity. Also shown in Figure 4 is the use of the DEFINE term.

MIS implementation and installation

The concepts, philosophies, and benefits of management information systems are now accepted by most areas of management. The greatest difficulty was gaining management's acceptance of the time and expenditure required for successful implementation and installation of an MIS system.

Early in 1966, efforts to establish a terminal-oriented management information system in the Poughkeepsie Laboratory were begun. Considerable systems analysis and programming experience of on-line terminal systems had been obtained with ATS. This helped to make management aware of the value of such systems, more knowledgeable about their functions, and more appreciative of the efforts to make them operational.


Security Code
File Name
Basic Format Terms
Selection Criteria
Control Term

```
XYZ63 PERS   NAME SER MGR HIRE/DATE DEPT WL D12,D19 JCODE EQ 8412 LIMIT 90.

Resulting Output:

  EMPLOYEE     SERIAL                       HIRE                JOB
    NAME        NUM.      MANAGER           DATE     DEPT       CODE

  Rogers, E    14368      J. Jones          01/64    D12        8412
  Evanson, J   23145      I. Brun           08/61    D18        8412
  Walters, R   41398      J. Jones          10/66    D12        8412
  ——
  ——
  Barker, I    98314      R. Martin         11/58    D12        8412

  Record count 73.
  End of Inquiry.
```

```
The DEFINE function allows the above inquiry to be reduced to a concise
inquiry statement when there is a periodic need for the data.

To set up this report the inquiry would be:

XYZ63 DEFINE REPORT/108 PERS NAME SER MGR HIRE/DATE DEPT WL D12,--- 90.

After MIS accepts the above definition, the basic inquiry is satisfied
with a simple request:

XYZ63 CALL REPORT/108.

The same output as shown above will result from this inquiry without
entering all terms and criteria involved.
```

Figure 4—MIS inquiry example

At that time a system called MICS (Management Information and Control System) was operational in the IBM DP Midwestern Region, and a follow-on System/360 effort was starting to convert it to System/360. The capabilities of this system were valuable to the Poughkeepsie Laboratory and made an appropriate starting point for moving into the on-line management information systems' world. The effort began by canvassing the "bread and butter" application areas of the laboratory such as purchasing, personnel, and finance. These areas had data processing as an integral part of their operating life, and they were constantly subjected to new reporting requirements and frequent statistical gyrations. Because of these two points, it was believed that MIS/360 could easily be applied to their problems and easily understood and appreciated by management.

Some people felt that top laboratory management should be addressed first, and that their support and active involvement in MIS/360 should be enlisted before applying it to operating levels of management. This approach was strenuously opposed for the following reasons:

1. Problems in the early stages of implementation and installation were anticipated, and it was felt that top management should not be burdened with them.
2. Initial problems may have caused opposition to MIS, making it difficult to revive interest at a later date.
3. Operating areas would provide the best exposures for MIS. It would be better utilized and more easily justified at this level.
4. The current state-of-the-art was more in line with operating level management, and further sophistication of MIS was needed before it could truly satisfy top management needs, directly.

Once the approach was decided upon, management became enthusiastic about MIS. "We like it, when can we get it?" In many instances, this enthusiasm cooled slightly when costs and installation time were given. Management enthusiasm waned even more with the realization that their assistance was vital to a successful MIS effort. The laboratory's ATS helped to overcome this resistance. It enables potential users to better evaluate the benefits and capabilities of MIS and to relate them to the solution of their specific information-gathering problems.

After this initial canvass, it was obvious that a system such as MIS/360 was needed in the laboratory and could be justified. An initial proposal was made to laboratory management showing areas of potential use, hardware requirements, and cost. It was tentatively accepted with the requirement that hardware needs be incorporated into the overall Computing Center equipment plan to ensure maximum equipment utilization at the lowest cost. Many passes were required before a meaningful equipment plan was developed and the required equipment ordered.

In parallel with equipment planning, applications analysts were assigned to areas where MIS had the greatest potential. They began detailed systems analysis to precisely define how MIS was to be used and what procedures and programs would be required to support it. This took more manpower than was originally anticipated. Each application area took approximately one man, full time for two to three months, to accomplish all tasks necessary to effectively implement MIS. These tasks included learning the area's mission; understanding its problems and needs; relating MIS capabilities to these needs; defining reports; making programming modifications to existing batch systems where required; preparing MIS table cards; ordering and installing terminals, data sets, and telephone lines; and educating the personnel in each area.

As more analysts and application areas became involved, the original organization plan became unworkable. A terminal systems group was established to coordinate all MIS efforts in conjunction with the other systems programming groups in the department. The group's responsibilities included:

1. Coordination of all MIS application areas.
2. MIS programming and support.
3. Control of MIS equipment ordering and installation.
4. Pre-analysis and feasibility studies of new applications.
5. Control of MIS control tables, editing, and storage allocation.
6. MIS user education.
7. MIS cost center control.

This relieved the applications analysts in the other systems groups from many time-consuming tasks not directly related to the customer's problem. This group was also responsible for an in-depth analysis and implementation of MIS applications not related to an existing batch processing system.

By February of 1968, there was a solid MIS installation plan with definite customer commitments. This plan called for a year-end target of 20 lines and 35 terminals. The IBM System/360, Model 50, was installed in March. By April 1968, the initial files were loaded and MIS became operational. (See Figure 5 for MIS Planning and Installation Phases.) In May, the first serious system problems arose. These were a combination of hardware and software difficulties and took almost three weeks to resolve. These problems were not unexpected; with a real-time terminal system start-up problems were anticipated. Since then only minor problems, which were quickly resolved, have marred system performance. Statistics show a steady growth in transaction load, increasing from 600 inquiries per week in early June to 2700 inquiries per week in October.

A vast amount of preliminary work allowed the original MIS installation timetable to be followed almost to the letter. Figure 6 shows actual installation dates for the various application areas; as shown, the year-end plan to install 20 lines with 35 terminals was substantially exceeded.

During the first six months of operation, installation target dates were slightly behind schedule. Within a one month period, the demand and commitments were such that not only were end-of-year objectives exceeded, but there was a queue of areas demanding MIS service. The implementation effort reversed. Instead of a struggle to develop the required MIS base, it was now an effort to install new applications within a reasonable time without exceeding system capacity. This was not unexpected. After MIS had started to prove itself and its capabilities were better understood, the demand went up accordingly. As a result, the efforts of the MIS system group were redirected. New plans were developed for increasing MIS capacity to satisfy the expanding needs of the laboratory.

User education was a crucial effort throughout the MIS implementation period. The objective was to formalize the education process so that all users received the same instruction and to accomplish this with minimum expenditure of Systems and Procedures manpower. The classes were conducted in three half-day sessions. Work periods and terminal operation time were integral parts of the sessions. Two people conducted each class, and the major portion of each session was followed by problem-solving work sessions. These sessions were the key to understanding MIS.

MIS applications

MIS/360 has been applied in various areas, including

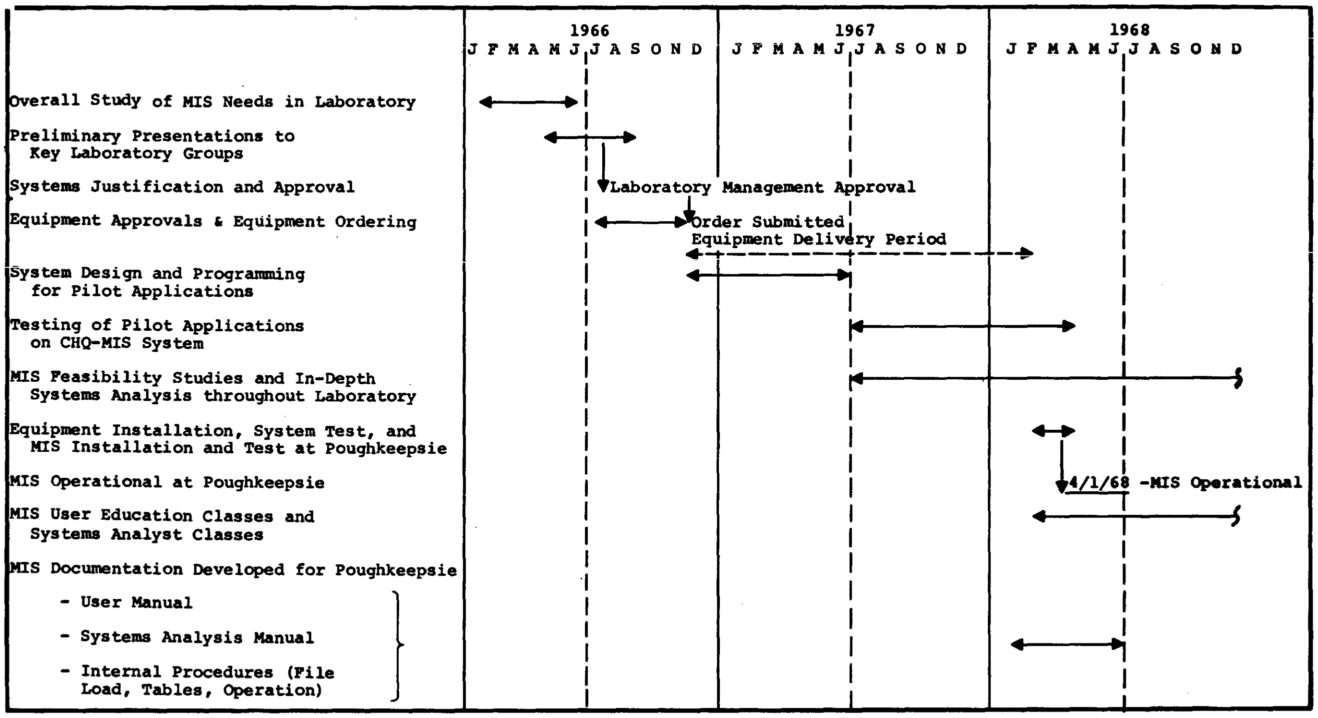


Figure 5—MIS planning, analysis, and installation schedule

APPLICATION	'68 Apr.	M	J	J	A	S	O	N	D	No.of Lines 12/31/68
Personnel & Recruiting	2	→	→	→	→	→	→	→	2	2
Purchasing	1	→	→	→	→	→	→	→	1	1
Computer Services	1	→	→	→	→	→	→	→	1	1
Facilities Services	1	→	→	→	→	→	→	→	1	1
S&P Reports Management	1	→	→	→	→	→	→	→	1	1
Financial Disbursement	1	→	→	→	→	→	→	→	1	1
Customs Systems Engineering		3	5	8	10	10	12	14	14	4 *
Engineering Education		1	→	→	→	→	→	2	2	1 *
Electronic Component Information System		1	→	→	2	4	→	→	5	3 *
P/S APAR Control		1	→	→	→	→	→	→	1	1
Document Control			2	→	→	→	→	→	2	2
Budgets/Analysis				1	→	→	→	→	1	1
Memory Products					1	→	→	→	1	1
Components Division						1	2	3	3	2 *
EIS Project Control							1	→	1	1
P/S System Test							1	→	1	1
TERMINALS	7	13	17	21	25	28	33	37	38	
LINES	7	13	15	16	19	21	24	24	24	24
# of Areas Using MIS	7	11	12	13	14	15	16	17	17	

NOTE: The asterisked applications involve MIS Networks. These MIS users have more than one terminal per line into the computer and as a result they share and contend for the lines assigned to them.

Figure 6—MIS application installation
April 1968 to January 1969
(MIS terminals operational)

programming, administrative, and engineering activities. This section discusses, in depth, the MIS application in the Custom Systems area and gives a brief description of other significant areas.

Custom systems

Custom Systems is responsible for evaluating, pricing, and engineering all special features to released IBM equipment within the Systems Development Division. These requests for products outside of the IBM product line are generated by a customer through a salesman. They result in a Request for Price Quotation (RPQ), which is directed to Custom Systems.

When RPQ's are submitted to Custom Systems, records are created for each RPQ. Activity against an RPQ causes an updating of the record. Data that are recorded include RPQ number; description; estimated and actual costs to develop, manufacture, and install orders placed against any RPQ; customer's name and the IBM machine-type. A system of batch-processing programs enables updating and control of the data base that contains pertinent data associated with all RPQ's. The master files that comprise the data base are periodically loaded on the MIS System. These files are:

The *RPQ statistical file* contains a record for each RPQ that is received in the Custom Systems Record Department for each Laboratory location. The file is in

RPQ number sequence. It is used to provide both statistical data and the status of a particular RPQ.

The *Laboratory order file* contains a record for each order and for the work in process at a Laboratory. It is in order-number sequence within RPQ number. The file contains such detailed order data as machine and model on order, scheduled shipment date, customer's name, and quantity ordered. The data in this file are used to analyze the order phase of RPQ processing as it pertains to each Laboratory.

The *SDD order file* contains a record of each open order for every RPQ, whether initiated by a laboratory or a regional office. The information is submitted to the Custom Systems Department in Harrison from the plant order departments. The data are on a year-to-date basis for the current year only. This file is used to analyze order information on an SDD basis.

The *RPQ price file* is a detailed file of priced RPQ's that are currently active. It is in machine type-model sequence within RPQ number. The data are customer, laboratory, and machine indicative. Information is available by RPQ number, customer number, branch office number, region or laboratory, and machine type.

The *RPQ cost file* contains a record for each RPQ that has a completed cost associated with it. Detailed cost estimates and actual cost data exist for those RPQ's that are priced and eventually ordered. The file is in machine type—model sequence within each RPQ number. This file is widely used by management for budgetary and other financial analyses.

The *RPQ reference file* contains a description of each priced RPQ that is active in the RPQ price file. It is maintained in RPQ number sequence. The data in this file describe the purpose and capabilities of the special features for IBM equipment.

The *RPQ error file* is a control file. Errors detected in entering and updating data are maintained for examination and correction by the appropriate Custom Systems Department. The corrections should be reflected in subsequent input transactions. The data in the file are current as of the most recent batch update.

The Custom Systems network of MIS terminals consists of four lines with 14 terminals contending for these lines. These terminals are located in the eight SDD Laboratories, and four Data Processing Division Regions, and at Division Headquarters in Harrison. This network allows sales personnel, Custom Systems engineering groups, and management to interrogate the system for RPQ descriptions, to analyze orders, to determine which System/360 systems are getting the most RPQ's, to see which customers are making the requests, and a host of additional information. The immediate availability of such information reduces distribution and mailing costs of reports and reference lists, enables sales personnel to be more responsive to customers, and reduces redundant engineering efforts at IBM Laboratories.

As an example of the use of MIS, assume that a Data Processing Division (DP) industry representative is selling a System/360, Model 65 to an airline. He wishes to know what RPQ's have been requested for that system by airline customers. He would interrogate the RPQ statistical file with the following inquiry to get his report.

Inquiry

```
XXXX RPQSTAT INDUSTRY CLASS
              EQ T1 SYSTEM EQ 2065
RPQ/NUMBER CUST/NAME B/O DEVICE.
  (NOTE: XXXX = Security Code,
                  B/O = Branch Office)
```

Report

RPQ NUMBER	CUST NAME	B/O	DEVICE NAME
A72097	ABC Airways	123	Cable Extension
E62077	Bird Airlines	456	Modify 7094 Emulator
G18999	Transport Co.	789	Console Change

This answer might lead him to an inquiry of the RPQ reference file to obtain a detailed description of the RPQ, and to the RPQ price file to determine its price. (Future plans for MIS will address a multifile interrogation capability with a single inquiry.)

To cite another example, suppose that Custom Systems management wishes to know the total rental for all RPQ's that are ordered, with a subtotal for each region. This inquiry would interrogate the SDD order file.

Inquiry

```
XXXX SDDORD CHARGE EQ R SUBTL/LH
                          REGION
CALC TOTAL/RENTAL 8.0 = QTY *
              UNIT/CHARGE NO/DTL.
```

In this inquiry, CHARGE is a field name and identifies which RPQ's are purchased or rented. SUBTL/LH indicates that subtotals are wanted for each regional office from low to high. CALC tells the system that the next field is not in the record and must be calculated. The 8.0 defines the TOTAL/RENTAL field size, with no decimals wanted. The TOTAL/RENTAL field is calculated by multiplying two defined fields, QTY and

UNIT/CHARGE. NO/DTL tells the system not to print detail lines.

Report

REGION	TOTAL/RENTAL
ERO	100,000 (fictitious numbers)
GEM	98,732
MWR	111,234
WRO	36,872
(TOTAL)	346,838

Summary description of MIS applications in process

Personnel records/recruiting

The Personnel Data System (PDS) has been used to create a 2000-character record containing significant information about each employee. Data fields such as promotion dates, salary information, year of degree, etc., can be addressed for an individual, or a group of individuals. More than 240 fields of information can be interrogated.

The Recruiting file contains all processing information about applicants, regardless of status, for the current recruiting year. When an individual applies for a professional position at IBM Poughkeepsie, a record is created. This record remains in the system until appropriate action has been taken and until recruiting year-end. Forty fields of data can be interrogated, including method and date of contact, test scores, departments visited, areas offered and accepted, acceptance date, etc. The Recruiting department uses this file in day-to-day operations, in the generation of weekly and monthly reports, and in year-end analysis work.

Financial disbursements

The initial files loaded are the Travel Accounting file and the Accounts Payable department's Request for Purchase Rejection file (RPR).

The Travel Accounting file enables management to analyze such travel information as who went where and how much did it cost. This file contains all closed records pertaining to employee travel advances and expenses, along with statistical and personnel information. Forty-four fields of data can be interrogated from the terminal.

The Accounts Payable file contains information about RPR's. RPR's can be analyzed, and recommendations about vendors can be given to management. Ten fields of data can be interrogated, such as purchase order number, ship date, vendor name and number, and disposition.

Facilities services

Initially, two files are utilized: (1) Plant Engineering Detail Project file, arranged by purchase order number within project number, and (2) Laboratory Facilities Services Summary Project file, arranged by project number. These files contain such information as estimated, committed, and actual costs for each facilities improvement or rearrangement project. The information is used for statistical analysis on projects, types of projects, vendors, etc. Seventy-two fields of data are available for interrogation.

Systems and procedures reports management

The Reports Management file includes key data about all reports prepared by SDD Poughkeepsie, such as report title, preparer, requestor, frequency, cost, number of readers, subject, and report number. Thirty-five fields are available for interrogation. A divisional and corporate system, utilizing the MIS system as a basic information retrieval tool, is planned for the future.

Electronic component information system

This MIS application provides a monitoring and reporting system used to track engineering change data, to determine whether objectives are being met, and to provide input to other systems, such as the logistics simulator. Events are recorded for each part number tied to a Request for Engineering Action (REA) or E/C. Typical data fields include origin and response dates, REA type, ship dates, machine group or part type impacted, recycle codes, E/C break-in dates, etc. Between 40 and 50 fields are available for interrogation.

This is a network-type application involving SDD Laboratories, Components Division, Systems Manufacturing Division, and Field Engineering, with three MIS lines allocated at this time.

Purchasing

The automated purchasing system includes MIS as an integral part of its overall design. Currently, the MIS file is a product of a series of System/360 batch-run programs. Future MIS capability will, for the most part, eliminate periodic creation of reports via the batch-run mode. Reports are created through MIS inquiries on a need-to-know and exception basis.

Two files are accessible through MIS/360: the Purchasing Master file that contains information about open purchase part orders and open interplant part orders; and a daily receiving file that contains records of a day's receiving activity.

Engineering education

A combined personnel-education data file for SDD Poughkeepsie employees serves as input to the data preparation program that creates the MIS education-load tape. The data preparation program is run on a quarterly basis. It extracts as output all management education courses, graduate work study courses, Syracuse University program courses, and all Engineering Education and Voluntary Education courses taken during the most recent four-year period. There are 62 fields available for interrogation. Some of the more pertinent ones are date of hire, educational level, educational objective, school codes, course number, course level, and course completion codes.

Document distribution and control

This area coordinates and controls the distribution of documents in SDD Poughkeepsie. Whenever a document is to be circulated, such pertinent information as those who should receive it and their respective addresses are obtained from the distribution master file. MIS loads the distribution master file and makes it available for terminal interrogation. Two terminals are installed in the area. One is used for information retrieval and limited updating; the other is equipped with a special pin-feed platen that allows MIS to print address labels for numerous distributions having a relatively small volume of labels.

Budgets/Analysis

This financial area handles budgetary control and analysis for the SDD Poughkeepsie Laboratory. Two files are loaded in MIS. One consists of the entire 12 month Budget master; the other is the Variance report master. The Budget master file makes the details of each area's budget available for terminal interrogation. The Variance report allows the financial area to analyze all areas for any variances (planned budget vs. actual expenditures) either on a current-month basis or on a year-to-date basis.

Applied programming analysis report (APAR) control

The file consists of the APAR master, with 33 fields available for interrogation. It includes APAR checkpoint dates, associated APAR codes, areas, etc., so that every phase of APAR activity is available for analysis and manipulation by MIS. An MIS network, for APAR control is a future possibility.

Memory products

This area uses an MIS file to monitor and control the memory engineering projects. The file contains a profile record for each unique stage or phase of the memory engineering project as it goes through development. Fifty-two data fields are available for interrogation. Some of the more pertinent fields are engineering project number, project phase code, estimated start and completion dates, revisions to start dates and completion dates, status codes, cycle time, and problem status pointers.

CONCLUSION

The MIS/360 System described in this paper is an integral part of the operating and management activities of the Poughkeepsie Systems Development Laboratory. Users of the system have found that it aids them in "doing business a better way." They can evaluate more things in more ways and in less time, allowing a better base for the decision-making process.

The system continues to grow—in numbers of lines, in numbers of terminals, and in numbers of inquiries, which is most indicative of how much a part of everyday operations MIS has become. It is intended that the system will grow as its use grows. Future plans call for a larger CPU and improved MIS software with additional capabilities. These activities must be continuous. As user demands upon the system increase, it must be made more responsive. It must exhibit higher degrees of sophistication in its capabilities. These requirements must be met if MIS is to continue to be a dynamic and valuable management tool.

ACKNOWLEDGMENTS

MIS program development was the responsibility of Mr. George Dath of Corporate Headquarters, with portions of the effort accomplished at SDD Kingston, SDD Poughkeepsie and FSD Huntsville.

During the installation of MIS at Poughkeepsie, the systems analysis and programming efforts were accomplished through the cooperative efforts of S&P personnel, computer operators and management, and the operating people involved within each application area.

At the inception of the project in 1966, the contributions of Mr. W. D. Timberlake and Mr. B. H. Matteson, Jr. (former managers of S&P and Computation Services) during the planning and justification cycle were particularly significant.

Computers and Congress

by EDMOND S. MESKO

Technical Information Services Company
College Park, Maryland

INTRODUCTION

The Instititute of Management Sciences (TIMS) and the Washington Operations Research Council (WORC) co-sponsored two meetings in the early months of 1968 in the Rayburn Building on Capitol Hill. In these meetings, these professional associations held what was probably the first dialogue between the Management Science community and Representatives of the U. S. Congress. The Representatives were Congressman Robert McClory of Illinois and Congressman F. Bradford Morse of Massachusetts.

The purpose of these meetings was to search for areas where Management Science techniques could come to the aid of the Legislative Branch of the Federal Government. Neither the members of the House of Representatives, who were the guest speakers at the meetings, nor discussants, nor the members of the audience had any recourse but to say, "YES! Management Science Techniques can help Congress!" But as suggested by Congressman Morse, the Management Science community must make the management science terminology more "fashionable" so the members of Congress will add it to their vocabularies, and must also "sell" Congress on the great potentials of Management Science technology.

Background

Congressional record

There have been several articles in the Congressional Record regarding ADP assistance to Congress. One article appeared on October 19, 1966.[1] This was the introduction of a bill by Congressman McClory of Illinois. The bill authorizes the Legislative Reference Service of the Library of Congress to make use of automatic data processing techniques and equipment in the performance of its function in support of the Congress. Congressman McClory noted the growing dilemma of Congressmen and their staffs to screen, sift and extract significant information from the ever increasing volume of data they receive.

Some of the very basic information that the Congressmen must know includes: status of current bills, legislative history of bills, schedule of committee hearings, budgetary data and facts and figures regarding everything from information about his constituents to information about unidentified flying objects.

Another article appeared in the January 30, 1967 Congressional Record.[2] In this article Congressman McClory introduced an abridged version of a study prepared by the Legislative Reference Service of the Library of Congress. The study was a review of various ways in which computers might be used to aid the Congress, Congress as a unit, each chamber of Congress, the committees of Congress, and the individual Congressman.

The third article in the Congressional Record was introduced by Congressman Tom Railsback on January 29, 1968.[3] The article is Congressman McClory's speech to the joint TIMS/ORSA meeting on January 17, 1968. In his speech Congressman McClory reviewed the need for ADP equipment to aid Congress. He also described a recently installed on-line terminal system which was installed in the American Law Division of the Legislative Reference Service of the Library of Congress. These terminals enable the Legislative Reference Service to store on magnetic tape descriptions of all bills and resolutions introduced in the 90th Congress. This data will be used to compile and list the "Digest of Public Bills". Eventually the system will enable a Congressman to retrieve any bill by number, title or word descriptors. When we realize that 26,000 bills and resolutions were introduced in the 89th Congress, we can begin to see one small area where Congress can be aided by ADP.

In his speech Congressman McClory reviewed his statement which he inserted in the October 19, 1966

Congressional Record. He also mentioned that the bill which he introduced in that Congressional Record still is pending.

Congressman McClory mentioned in his speech a fact that was surprising to me. He noted that Congress in 1967 appropriated over $1.2 billion for the 3,000 computers in use by the departments of the executive branch of the federal government. However, the Congress has refused to appropriate one-thousandth of that figure to equip itself with an ADP capability.

Studies of Congress

There have been several studies and books regarding Congress and Congressional reforms. I will discuss only those areas regarding ADP. The studies are: the Arthur D. Little Study that was sponsored by the National Broadcasting Company, the results of which were made into a television Special;[4] the Twelve Studies of the Organization of Congress conducted under the auspices of the American Enterprise Institute for Public Policy Research;[5] the Dartmouth College study, which was directed only to the House of Representatives;[6] We Propose: A Modern Congress, which is a series of articles by members of Congress;[7] and the Report of the 1965 Joint Committee on the Organization of the Congress, which is now represented in bills before the Senate[8] and the House.[9]

The common thread running through all the studies is the problem of obtaining timely, accurate, complete and relevant information for decision making. During the Dartmouth study,[6] four out of five Congressmen said that the lack of information and complexity of decision making were the major problems preventing Congress from performing more effectively. This is true for the individual Congressman as well as for the Committees of Congress. The Dartmouth study was in the form of 32 reform proposals drawn up by the study team. The proposals were developed after a search of the extensive literature written about the Congress. The team members then asked a total of 116 members of Congress what their thoughts were regarding the 32 proposals.

The need for better information was also mentioned in the Report of the 1965 Joint Committee on the Organization of the Congress.[6] The report urged the use of automatic data processing to provide expanded budget information to members of Congress to aid in Congressional fiscal control and budget evaluation. The result of this report has been formed into bills now pending in Congress.[8,9] I will discuss this pending legislation in a later section. This report was an internal study of Congress. The study was co-chaired by Senator Mike Monroney and Representative Ray Madden.

The need for a computer to better analyze the budget, as well as the hundreds of other subjects, is also mentioned by David Brinkley, the NBC correspondent, in the introduction of the book, "Congress Needs Help."[4] One of the findings by the Arthur D. Little study team was that Congress does not take advantage of automatic data processing equipment to facilitate its work. Because of the massive volume of data input to Congressmen and Committees, it is only natural to turn to high speed, large capacity computers. The conclusions and recommendations of the study call for the use of the computer to manipulate the data into usable information.

The American Enterprise Institute for Public Policy Research is a nonpartisan research and educational organization. The purpose of AEI is to assist legislators and educators with studies of current issues of national significance. AEI compiled twelve studies of the Organization of Congress into the book, "Congress: The First Branch of Government."[5]

In the study, *Availability of Information for Congressional Operations*,[5] it was found that the complexity of decision making and the lack of information are the difficulties that were most frequently cited by the Members of Congress. In the study, the *Committees in a Revitalized Congress*,[5] it was noted that the information problem is not one of scarcity of information, but an abundance of information, most of which remains unassimilated and undigested. The study, *Information Systems for Congress*,[5] advocates the development of automated information processing systems to provide the information for legislative decision making.

ADP applications to Congress

Before we discuss how ADP can be applied to Congress, let us first categorize the working parts of Congress.[10] The functional areas of Congress can be divided into five parts:

1. Congress as a Unit
2. Each Chamber of Congress
3. Committees of Congress
4. Individual Congressmen
5. Political Parties

Within some of the functional areas there are two kinds of information that is required: legislative information and administrative information.[11]

The legislative information can be divided into information for current problems and information that could be relevant to future areas of concern.

The legislative information applicable to current problems and the legislative information being compiled for future matters can be divided and ordered by subject.

Let us now look more closely at the functional areas of Congress and begin to determine their information needs.[5]

Congress as a unit

This includes both Houses of Congress. Certain kinds of information is relevant to both the House of Representatives and the Senate. A centralized data bank should provide:

a. Legislative Information

1. *Status of Pending Legislation.* Considering that there are about 26,000 bills and resolutions submitted for action in each Congress, we can see that there is a need for a central file of this legislation. At this time there is no one place where a Congressman can easily find information regarding pending legislation.

   A Congressman should be able to have access to a centralized file of all legislation introduced into either House of Congress. He should be able to search for this information by the number assigned to the bill or resolution, or by subject. To go one step further, each Congressman could have his "interest profile" stored in the computer system and have this "interest profile" automatically search each piece of proposed legislation that enters the data bank. The "interest profile" would represent the personal interest of each Congressman. When the key words of the "interest profile" would match with the key words of the bill or resolution, an abstract of the bill or resolution would automatically be sent to the Congressman as a printout or output on his own terminal in his office. Included with each bill or resolution should be pertinent information such as the name and number of the bill, the sponsor, the content of the bill, related bills in the House or Senate, past legislation regarding both bills and laws germane to the current bill, and status of action on the bill.
2. *Lobbyist Activity Information.* Lobbyists are one of the prime sources of information for Congressmen. All lobbyists are required to register either with the Clerk of the House or the Secretary of the Senate. These lists are then published each quarter in the Congressional Record. However, they do not provide much information to the members of Congress.

   To assist the members of Congress it would be beneficial to record all lobbyists and related information in a central data bank. A Congressman should be able to search a data base through a remote terminal to determine if an individual is a registered lobbyist, his employer, the legislation he is concerned with, total sum of contributions he receives and the source, his past history and technical background, including his speeches and publications or editorials.
3. *Access to the Legislative Reference Service.* The ability of being able to access the Library of Congress' imaginary computerized data banks through the Legislative Reference Service (LSR) is potentially a very powerful tool. The Library of Congress is a vast storehouse of information. While the potential is great, the implementing of the system will not be easy.

   However, to review a point raised in the TIMS/WORC meeting, the American Law Division of the Legislative Reference Service of the Library of Congress has installed online terminals which enable LRS to enter and store on magnetic tape descriptions of all bills and resolutions introduced into Congress. Eventually a Congressman will be able to recall a bill by number, title or word description.
4. *Legal Information.* The University of Pittsburgh Health Law Center has recorded on magnetic tape the entire U.S. Code of Laws. It is possible to search the tape and select all the laws within a given subject and also it is possible to find laws pertaining to a particular subject but entered under different headings. Included in this system are the complete codes of several states, the U.S. Supreme Court Decisions since 1950, the Internal Revenue Code and Regulations, as well as other legislative and court information.

   Other legal information now in ADP form includes the Department of Defense sponsored Project LITE (Legal Information Through Electronics), legal information in several other Executive Branch agencies, and other legal information held by states, private organizations and several U.S. universities.

b. Administrative Information
   1. *Index-Catalog of Congressional Documents.* This could include all the Congressional documents published in either House of Congress. An example could be a listing by subject or related category of all the published committee hearings.
   2. *Congressional Employee Payroll.*
   3. *Legislative Telephone Directory.*

Each chamber of Congress

The information required here would be relevant either to the members of the House or Senate, but not to both.

a. Legislative Information
   1. *Location of Bills.* A Congressman in the House or Senate should be able to locate a bill that was introduced in his Chamber and also the status of the bill. He should be able to find out the history of action taken on the bill, whether it is in Committee or not, amendments to the bill, Committee votes, floor votes, scheduling for future action, and sponsors of the bill.
      The bill should be able to be retrieved by bill number, sponsor or subject.
   2. *Vote Information.* When the voting bell sounds in a Congressman's office, he must go immediately to his chamber to vote on a subject about which he may know very little. Currently the chamber based information regarding a vote usually comes from a colleague on the floor who knows something about the subject or else even by the doorkeeper. With a terminal in his office, the Congressman could be able to get an abstract of the bill on which he is being mustered to vote. This could give the bill number, sponsor and legislative history, and pro and con arguments.
   3. *Automated Voting.* Automated voting is now done in several foreign countries and in some of our states. However, automated voting is more a political problem than a technical problem. I will not summarize the pro's and con's, but merely say that it is technically very feasible.

Committees of Congress

The Committees of the House and Senate conduct the bulk of the legislative work in Congress.

a. Legislative Information
   1. *History of Committee Action.* Each Committee should have a history of all the bills that fall within its jurisdiction. The bills should be sorted by subject and should include related information such as the sponsor, the Congress in which they were introduced, the bill's provisions, whether or not any hearings were held, record of information supporting or opposing the bill, action by the other chamber on similar bill or bills and whether the bill becomes law.
   2. *Appropriation Information.* This is the area in which there is the greatest need for an information system. There is no lacking of appropriation data for both current and past expenditures. However, correlating this data into usable information is still a problem. This is not only true for the budget review within a single committee, but there is also a great need to have a cross-committee review of the budget. What is assigned to one committee can be directly related to a matter in another committee. The introduction of the Planning-Programming-Budgeting System (PPBS) may provide the government with an added impetus to convince the Congress that a computer could be used as a tool to aid in the review of the budget. The appropriation information should include statistics on past and projected budgetary expenditures for each agency of government. The information should include all appropriation pending for an entire program and not just the funding for the individual segments.
      Unfortunately, getting an across-the-board review of the budget is another area where the political problems of developing such a system are equal in scope to the technical problems.
   3. *Congressional Overview.* One of the prime functions of Congress is to overview executive agencies. For example, Congressional overview of the Space program is a function of the Senate Committee on Aeronoutical and Space Sciences and the House Committee on Science and Aeronautics. These committees should have the information to examine appropriations for the Space program and to correlate planned events and

proposed expenses with historically related data.

4. *Subjects Under Committee Jurisdiction.* Each committee is charged with maintaining an expertise in specific areas. These particular areas are defined in the "Rules of the House of Representatives" and the "Senate Manual." The committee members should have access to significant and pertinent information relating to each of the subject areas and should have the ability to browse through this information such as we do when we look in a library's catalogs.

b. Administrative Information
A schedule of all committee meetings should be maintained to enable committee members to better plan their time. Members of the House and Senate may belong to more than one committee.

Individual Congressmen

Congressman Morse, in his discussion before the joint TIMS/WORC meeting in the Rayburn Building, said that his workload is about 75 to 80 hours per week. It would be crystal ball gazing to prophesy that computers will reduce his workload to 60 to 65 hours per week. However, there is no doubt that ADP could help Congressman Morse make more efficient use of his 75 to 80-hour work week.

a. Legislative Information
   1. *Personal Information.* Each Congressman's office should be equipped with an on-line system to handle his own personal information file. The system should fit the individual Congressman's needs. Each Congressman's interest as well as style of operations vary. The file could be divided between the Congressman's special long-term interests and current legislative obligations.
   The long-term interests could be public works projects within his district or possibly a strong interest in a national matter such as water or air or noise pollution.
   The file of current legislative matters could include his voting record on a particular subject, a summary of pending legislation, a summary of his constituents' attitudes, and a recall of his public philosophy that he expressed in speeches, statements to the press and in various publications.
   2. *Reference File.* Each Congressman must wade through a mountain of data everyday. He can scan some of the reading material, but he must read other material sentence by sentence. A Congressman should have a system that could retrieve information based on his own key words. This system could either present a listing of the relevant documents or else it could project the written page on a screen.

b. Administrative Information
   1. *Re-election Information.* The Congressman's prime issue is to get re-elected. A Congressman must know the total amount of campaign contributions, a list of the donors and the amounts they donated, the donors' interests, the amount spent in a re-election campaign, and the manner in which it was spent. He must also know the voting blocks within his district. These include the unions, business leaders, civic leaders and the interest groups, such as a farmer's association. The system should also be able to analyze polling data and election returns.
   2. *Constitutent Data.* Each Congressman should also have access to the names and addresses and other relevant information to each of his constituents.
   As you can see, this administrative information deals with a very sensitive subject. Many people feel that this would lead to the "Big Brother" state described in George Orwell's *1984*. Too much information in the hands of the wrong people is bad. Too little information in the hands of the right people is bad. It would be a difficult task to justify every bit of information that went into such a data bank. However, in the near future this is precisely what must be done. Technical potential must be tempered and guided by social conscience.

Political parties

Each of the political parties has National and Congressional Committees. The Political (Party) Committees of Congress include both the House and Senate. Each of these Committees has a specific purpose and are, therefore, interested in specific information. Some of the committees for both Republicans and Democrats are the Policy Committee, Steering Committee, Personnel Committee and Campaign Committee. Information required, therefore varies from campaign information to the planning of political strategy in the House or Senate.

The National Committees could include policy information relating to party objectives and policies; information by states or areas of the country; or information by categories such as the Space program, or air or water pollution, or information on the overall policy toward cities. Other information could include the opposition party's policies and objectives and the arguments against those policies and objectives.

Other information could include administrative matters. This would include voting information such as campaign planning and funding, the names and addresses of state and local leaders, the opposition party's strong and weak areas, and the policies and political backgrounds of voting blocks possibly divided by economic strata, ethnic groups, and/or geographic areas.

Congressional committees

Any of the above-mentioned candidates for ADP applications could be discussed more deeply. However, we will discuss only the problems relating to the standing committees of the House and the Senate. There are 20 standing committees of the House of Representatives[12] and 16 standing committees of the Senate.[13] The Senate Manual and the Rules of the House of Representatives list the standing committees and the power and duties of each of the standing committees. The committees were last restructured in 1946 to better delineate the jurisdictional areas of each of the committees and to better parallel the executive agencies.

Whether or not this precise delineation of responsibility and attempt to parallel the executive branch is satisfactory will not be discussed in this paper. However, this situation is mentioned because it indirectly leads to an example of the problem caused by the current committee structure and how this problem could be somewhat alleviated through the use of ADP's ability to provide horizontal integration of a similar subject.

Horizontal integration with ADP

The House of Representatives has a Committee on Education and Labor. The Rules of the House of Representatives lists and defines one of the jurisdictional areas of this committee to include "Measures relating to education or labor generally." The Senate has a Committee on Labor and Public Welfare. The Senate Manual defines one of the jurisdictionals areas of this committee to include "Measures relating to education, labor, or public welfare generally." Because these powers and duties are specifically listed in the rules books of both the Senate and the House, we would assume that all education matters are handled by these two committees. However, this is not the case. It is revealed in the book, "Congress Needs Help"[4] that about thirteen different Senate Committees, fifteen House Committees, and one Joint Committee considered education bills in the 89th Congress. Congressmen would have to go to 29 different committees to get information regarding education bills in the 89th Congress. This is an excellent case for justifying a retrieval system using key words.

Currently, each of the committees has a committee calendar. The Committee on Education and Labor publishes their calendar monthly on a cumulative basis for each Congress. The calendar is organized three ways: sequentially by bill number, by author, and by subject. The committee staff updates a working copy daily with the same information they provide to the Legislative Reference Service for the Daily Digest. The Daily Digest is not cumulative.

A staff member of the Committee on Education and Labor explained that it was no trouble in his office to find information in the updated office copy of the Education and Labor Calendar regarding any education bill in that committee. However, the only way for him to get information regarding other education bills in other committees would be to look at the working copy of each committee's calendar. This would be a very time-consuming project. He said that there is a need for a Daily cumulative House Calendar that would include the status of all bills whether they are in committees or not. This House Calendar could be organized in the same manner as is the Education and Labor Calendars. He said that this House Calendar would be valuable in three ways: time saving for staff members, more rapid response of Congressmen to constituents, and more adequate voting information for Congressmen.

This would not be a complex problem to solve. The data are already there. It would be a matter of compiling the data in one location and organizing the data in an integrated data base. This situation would fit in very well with an on-line information retrieval system using key words.

History of committee action

One of the prime sources of information for committees is the committee hearing. The word for word transcriptions of hearings are published in a book form. Most of these transcriptions get to be about the size of a book. And there is more than one hearing in a committee. In the House Committee on Armed Services in the 89th Congress there were 102 printed hearings and special reports, containing 11,848 pages.[14]

There were also 396 meetings by the full committee and its subcommittees.

It would be beneficial to have an on-line system through which summaries of these hearings could be reviewed using key words. In this manner, the members of the committee who were at the hearing could refresh their memories and new committee members or other Congressmen who were not at the hearings could get a concise, accurate review of the hearings without having to work their way through a great deal of extraneous words.

A spokesman for the Committee on Education and Labor said that the current information retrieval system is to remember what was said at the committee hearing or else to read the 300 or 500 page transcript.

Subject under committee jurisdiction

While there are many areas where ADP can assist the committees of Congress, we must not assume that all the information for any one subject will always be found in the subject file. Staff members of the Senate Aeronautical and Space Committee get bits and pieces of information from the Bureau of Census, NASA, HEW, Department of Transportation, the National Academy of Sciences, the State Department and the airplane manufacturers.

One of the staff members believes that there is no place for a computer in a committee. The reason given is that the sources of information are too scattered and varied and too unstructured to be satisfactorily assimilated and compiled into a unified format. Much of the information is needed on an immediate basis. For example, a member of the committee requests in the morning material for a speech to be given in the afternoon. The staff members do not have the information, but they do know who to call and the right question to ask.

It is my feeling that the members of the committee staff are not afraid that the computer will replace them, but rather are wary that a computer system is not flexible or dynamic enough to receive, process and output information within the constraints imposed by the function and purpose of the committee. To an extent this is true. However, any large organization has routine input and historical data that can be structured and processed and presented in a logical and varied format. When looked at in this manner, the Senate Committee seems very similar to any organization.

ADP as a tool for committees

The staff directors and the chief clerks must be made to believe that a computer is only a tool. It does not dilute the powers of the committee members or staff directors or chief clerks. It should be made to enhance their power. It should enable them to better organize and structure the information they already have and to present the information in a timely and orderly and concise manner.

A bill to improve the operation of the legislative branch of the Federal Government

A bill to improve the Operation of the Legislative Branch of the Federal Government was introduced in the House[9] and Senate[8] of the 90th Congress. The bill is the resulting product of the Special Committee on the Organization of the Congress. There are numerous mentions of areas where ADP can be applied to aid the Congress.

One area specifies a data processing system for budgetary and fiscal information and data. The bill states that "The Comptroller General of the United States, the Secretary of the Treasury, and the Director of the Bureau of the Budget shall develop, establish, and maintain, for use by all Federal Agencies, a standardized information and data processing system for budgetary and fiscal data." The Comptroller General is also required to specify the location and nature of data relating to various Federal agencies' programs, activities, receipts and expenditures. It is also specified that the Comptroller General establish within the General Accounting Office the data processing systems. The Comptroller General is also authorized the funds to obtain the services of individual experts and consultants for assistance.

In another part of the bill, each standing committee of the Senate or House is authorized to contract the services of consultants or organizations to make studies or advise the committee with respect to any matter within the committee's jurisdiction.

The Director of the Legislative Research Service, i.e., the bill proposes changing the name of the Legislative Reference Service to Legislative Research Service, is also authorized to procure the services of individual experts or consultants learned in particular fields of knowledge. Also in order to facilitate its performance, the Legislative Research Service may (1) prepare information for machine processing, (2) process information by machine, and (3) prepare information for presentation by machine. The Service has also authorized the funds to acquire automatic data processing equipment to implement the specified work.

The bill also would establish for the Congress an Office of Placement and Office Management which would be supervised by the House Committee on House Administration and the Senate Committee on Rules and Administration. The Office of Placement and Office Management would maintain for the entire Congress, a list of private Management concerns capable of rendering studies regarding improving the efficiency of Congressional operations.

The Senate version of the bill also proposes the establishment of a Joint Committee on Congressional Operations. One of the functions of this Joint Committee is to make a continuing study of automatic data processing and information retrieval systems for use in the House and Senate and to recommend the implementation of these systems. To assist in this matter the Joint Committee is authorized to procure the services of consultants or organizations knowledgeable in the particular areas.

While these bills were not passed, there is general approval and intent among Congressmen to submit other bills to pursue the goal of adding automated data processing support to meet congressional needs.

CONCLUSION

I have presented to you the current thought on how ADP can aid the Congress. The needs and solutions have been mentioned by members of Congress in books and official publications of the Congress; they have been specified in a joint committee study of the Congress, and they are now documented in a bill introduced in the 91st Congress.

This bill is a door. Beyond this door is a room of boundless dimensions limited only by our imagination, technical knowhow and salesmanship. I hope I have done my part in presenting this door and a picture of what is behind the door.

ACKNOWLEDGMENTS

I wish to thank Robert Chartrand, the Information Science Specialist of the Legislative Reference Service of the Library of Congress, for his encouragement and for suggesting various references that have substantially added to my paper.

REFERENCES

1 R McCLORY
An automatic data processing facility to support the Congress
Congressional Record, Washington October 19 1966

2 R McCLORY
Congressman McClory recommends automatic data processing study
Congressional Record Washington January 30 1967

3 T RAILSBACK
Congressman McClory suggests computer uses for Congress
Congressional Record Washington January 29 1968

4 P DONHAM R J FAHEY
Congress needs help
Random House Inc New York 1966 Chapter 6 7

5 *Congress: The first branch of government, twelve studies of the organization of Congress*
American Enterprise Institute for Public Policy Research Washington D C 1966

6 R H DAVIDSON D M KOVENOCK M K O'LEARY
Congress in crises: Politics and congressional reform
Wadsworth Publishing Co Inc Belmont California
Hawthorn Books Inc New York 1966 Chapter 1 4

7 *We propose: A modern Congress, selected proposals by the House Republican Task Force on Congressional reform and minority staffing*
McGraw-Hill Book Co New York 1966 Chapter 20 21

8 Senator Monroney, from the Special Committee on the Organization of the Congress
A Bill 90th Congress 1st Session S. 355

9 Congressman Bolling, "Legislative Reorganization Act of 1967."
A Bill 90th Congress 1st Session H R 10748

10 R L CHARTRAND
Computer-oriented information for the U. S. Congress
Law and Computer Technology Vol 1 No 2 February 1968

11 R L CHARTRAND
Congress seeks a systems approach
Datamation May 1968

12 L DESCHLER
Rules of the House of Representatives, Ninetieth Congress

13 G F HARRISON J P CODER
Senate manual, Ninetieth Congress

14 *Organization Meeting of House Committee on Armed Services, Ninetieth Congress*
Washington D C February 1 1976

15 R L CHARTRAND K JANDA M HUGO ED
Information support, program budgeting, and the Congress
New York Spartan Books 1968

Automatic checkout of small computers

by MARVIN S. HOROVITZ

Digital Equipment Corporation
Maynard, Massachusetts

INTRODUCTION

The testing of a computer may take on many forms, one form is as follows. The computer may be tested in a segmented manner.

IC
MODULE
POWER SUPPLY
MEMORY
WIRING
CP TEST (PHASE 1)
ENVIRONMENTAL (PHASE 2)
ACCEPTANCE (PHASE 2)

Each of the above segments makes use of either a dedicated logic exerciser or a dedicated computer or a time allocated computer system. The IC, MODULE, POWER SUPPLY, MEMORY and WIRING are assembled and tested prior to the entrance of Phase 1 and Phase 2 testing. This paper deals only with Phase 1 and 2 testing.

Phase 1 testing

The objective of Phase 1 is to exercise a computer under test and check the resulting status of that machine. In designing a test system of this nature a prime concern is finding any malfunction that may exist within the computer under test. Each mode of every instruction, Memory System, I/O Interface and Manual Control must be completely validated. To accomplish this, initial conditions must be established. The computer under test (PDP-8/I) must then be stimulated and the results of the operation monitored for comparison with anticipated data. If an error is detected, the test system must have the capability of reporting the error and repeating the test the computer failed to perform. The system is designed so that the computer under test can never impede the repeated cycling of any test.

System hardware

To maintain continuous testing of the PDP-8/I at its normal clock speed would require a test system with enormous capabilities. A more practical approach is to design a system that will test the computer at its normal clock speed, but only for short increments of time. The test system hardware can be designed so that when properly preconditioned by the Test System Software, it will cause the PDP-8/I to function at normal speed for the specified interval with no further stimulus. Time between tests is used for checking the results of the previous test and to set up for the next. This time is large compared to the actual time spent testing. This enabled us to use the PDP-8 as the test systems computer. Design is simplified when the test systems computer and the computer under test are the same type machine. Data manipulation by both hardware and software is facilitated when both machines use the same word size. Also, a computer of the same type is usually available in the production area and need not be a permanent part of the test system.

The test system must have an input peripheral device for program loading and an output device for reporting errors. This test system uses a disk for input and a printer for output.

There is a special interface which acts as a control and sensing device to the computer under test and is completely program controlled by the test system. With the appropriate commands, the tester will perform the following test functions in the PDP-8/I under test through the special interface.

1. Manipulate any key or switch logic.
2. Simulate memory read operations by transferring data into the memory buffer at the proper time.
3. Sense memory read and write drive pulses produced in the memory address decoding circuits.

4. Sense memory inhibit drive pulses produced during the memory write cycle.
5. Sense the ON or OFF state of all indicator light source functions.
6. Sense the one or zero state of all timing clocks such that any timing state or pulse can be activated at will.
7. Control the PDP-8/I clock speed.

Figure 1 shows entire Phase 1 Test System Flow Chart and Figure 1/P shows the actual test station.

The PDP-8/I computer under test

Most central processors are equipped with an operator's console which uses switches, keys, and indicator lights. Generally, connection of this console is made to the central processor logic panel with plug-in type cables. While the PDP-8/I is being tested by the system, the operator's console is not used and these cable connections are made to the test system. All manual switch controls are operated remotely by the test system. The Test System program can simulate any of the actions of a console operator from starting and stopping the processor to observing the indicator lights.

For control of timing, appropriate modules are removed and timing control lines are cabled in from the test system. Control is such that the PDP-8/I may be run at any of several speeds from faster than normal to pulse by pulse.

The input-output peripheral buss cables of the PDP-8/I are connected to the test system. This affords an extensive test for the I/O interface in that the initiated peripheral commands can be monitored, and the response to peripheral initiated actions can be sensed by the test system.

During the initial stages of testing, the computer is exercised without a memory stack. Cables from the vacant memory area in the computer logic panel to the test system enable the system to monitor memory read, write, and inhibit pulses and simulate memory core changes on the sense amplifier input lines.

When some confidence has been established in the basic processor control logic and the memory control circuitry, the pre-tested memory stack and memory sense amplifiers may be inserted and more advanced tests carried on. Various combinations of machine instructions can be loaded into the PDP-8/I memory to form test loops that can be initiated and stepped through much as an operator would do at the operator's console. These program loops can be started and run at normal machine speed. Any program can be loaded and run in this manner which duplicates the condition of the program running in a free standing machine. However, in the event of a program wipe-out, due to a machine malfunction, a programmed high-speed reload and restart enables the system operator to observe repeated failures on an oscilloscope. This continuous cycling feature is not possible on a free standing machine when a wipe-out failure occurs.

System software

The system hardware provides many avenues through which the programmer may channel his efforts in attempting to find all existing problems in the computer under test. Two general approaches will be considered here, one being diagnostic in nature, the other being a specification check.

The specification check consists of outlining in detail every particular operation that the PDP-8/I is designed to perform. A series of tests is then written using various initial conditions and operands for each operation. Each test is provided with several check points at which the test system monitors the state of every available status function. Every status at each check point has been predicted by the programmer and is stored in internal tables of the test system program. If the monitored status is wrong, the comparison of the anticipated status with the monitored status results in an error report at the test systems printer.

The approach discussed above does not anticipate a failure in any particular hardware area at any time, nor does it attempt to exercise one area of the hardware

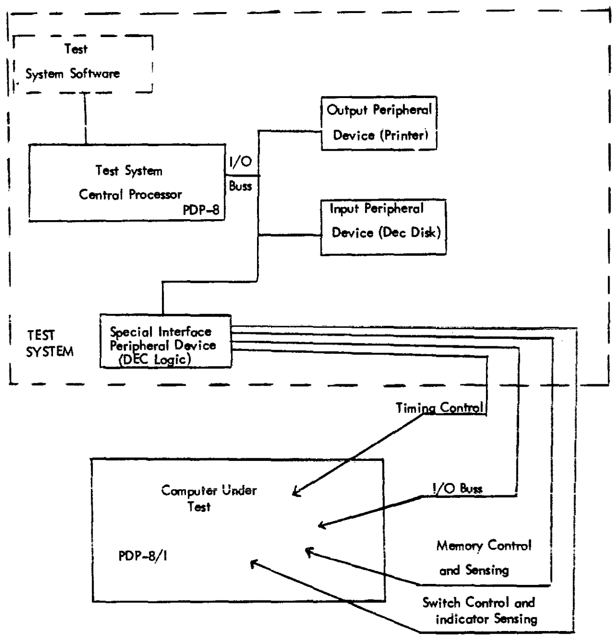


Figure 1—Flow chart of Phase 1 test system

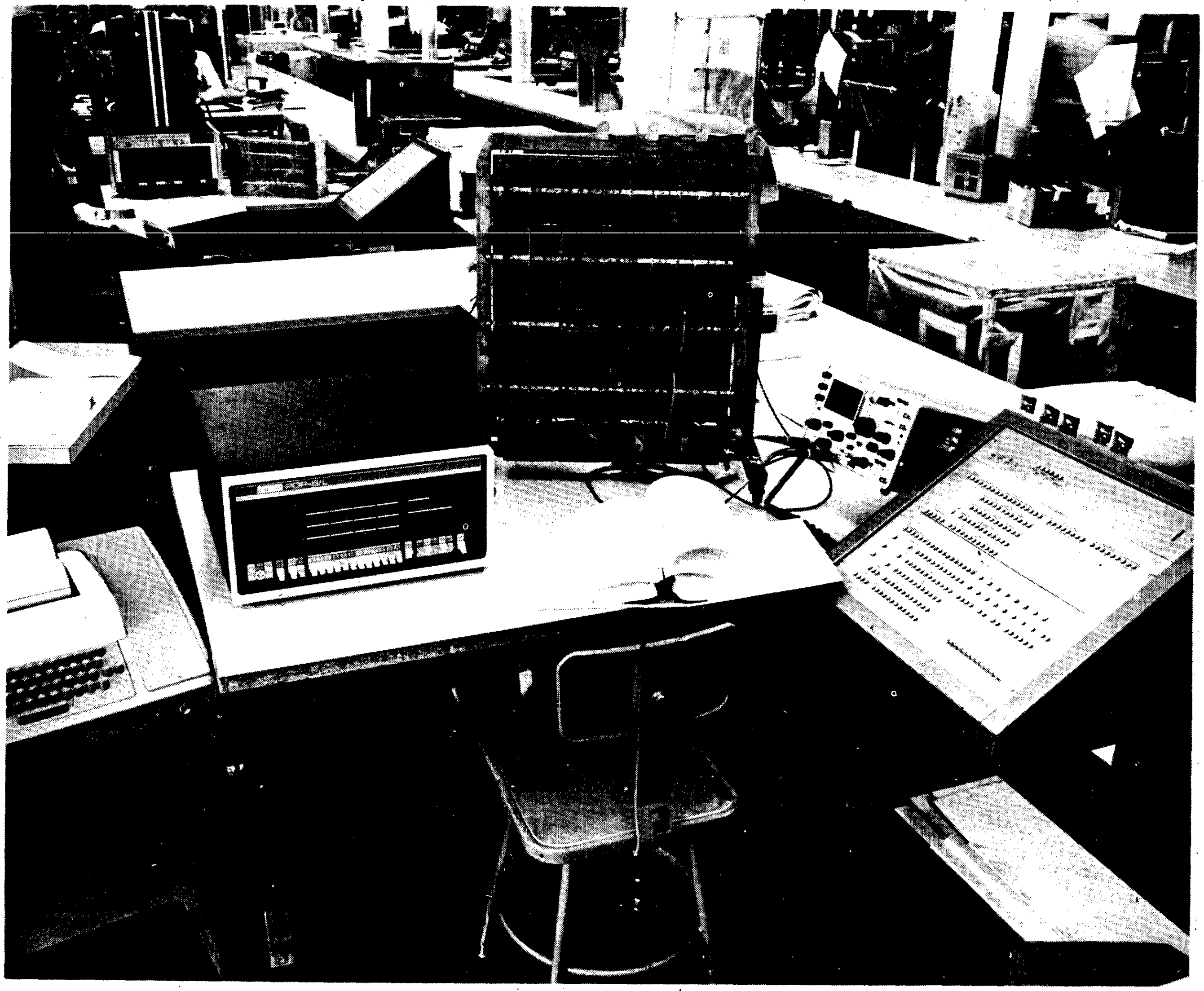


Figure 1/P—Actual Phase 1 test system

more than another, during a selected group of tests. It does, however, cause the PDP-8/I to peiform its specified operations under various conditions and checks for any sign of a malfunction anywhere in the computer during all the tests. In contrast to this specification check, the second software approach is aimed at validating isolated pieces of hardware. The building block method is employed where previously checked logic is used to test other areas.

As individual pieces of hardware are tested in small increments, the table of known functioning logic grows. Validated logic is considered sound when a new test is performed. By analyzing the results of many tests, a failure can be isolated to within a few logic modules. Using the proposed system allows the programmer to expand the set of tests whose results may be used to predict failing modules. This expansion results from the fact that the system can cause the computer to perform all of its specified functions plus many more, such as executing time pulses out of sequence, causing program interrupts at any specific time; stopping the processor clock at any points, etc. In actuality the building block tests are first used and then followed by the specification tests. The Memory System is then integrated with the computers logic. Memory integration tests include exercising memory with simple test patterns through complex test patterns. This is accomplished by using the PDP-8/I data break facility. (Direct memory access.) We now have, for all practical purposes, integrated the memory into the computer under test. The final test is to load small test routines directly into memory and run them at machine speed.

The results of these tests are constantly monitored by the test system. Error control is still handled by the test system. Ultimately through this method, the entire PDP-8/I logic and its memory system are completely tested and a complete computer is born.

Documentation

The more the test technician knows about the tests being made and the expected responses, the easier it will be for him to use the test system effectively in problem solving. For this reason, an extensive set of documents is recommended. This includes a separate timing and print out chart for each test programmed. These charts will list each test system action and the anticipated computer function monitored.

The system error reports will inform the operator of the failing test number, a failing subdivision within that test, the status that is wrong, and what that status should be. With this information the operator can turn to the print out chart for the failing test to get a composite picture of how the test is carried out. Listed, he will see the predicted status of every computer function monitored for evey subdivision of the test. With the entire test outlined before him and the problem area pin-pointed, he can formulate his plan of attack, which

TELETYPE PRINTOUT's

Printout example:	0001 S GB G0001 B0201				
Printout meaning:	No. of Test	State of 8/I Timing	Group Indicators	Correct Group Indicators	Defective Group Indication
	0001	S	GB	G0001	B0201

Number of tests = test which fault occurs in
State of 8I timing = time or cycle of 8I which test fails in
Group Indicators = registers, states, or significant pulses where test problem occurs
Correct group indication = correct contents of group indicator
Defective group indication = the actual group indication (wrong indication)

GROUP INDICATORS

MA - memory address	FR - fetch register
PC - program counter	DR - defer register
MB - memory buffer	ER - execute register
AC - accumulator	TR - transfer register
GA - Group A	BI - buffered MB bits
GB - Group B	SC - step counter
MS - major states	BC - buffered AC

Figure 2—Error message explanation group indicators

GROUP INDICATORS

MA	PC	MB	AC	GA	GB	MS	FR	DR	ER	TR	BI	SC	BC	BIT #
MA Ø	PC Ø	MB Ø	AC Ø	LINC	EXT. MA Ø	AND	FR BIT Ø	DR BIT Ø	ER BIT Ø	IOP 1	BMB Ø (1)	BUFF. BRK.	BAC Ø (1)	0
				PARITY FMR	EXT. MA 1	TAD				IOP 2		BUFF. RUN		1
				PARITY DMR	EXT. MA 2	ISZ				IOP 4				2
				PARITY EMR	$\overline{\text{BMB 3}}$	DCA				ADDR. ACC				3
				W. C.	$\overline{\text{BMB 4}}$	JMS				W. C. OVER FLOW				4
				C. A.	$\overline{\text{BMB 5}}$	JMP				BT 2A				5
				D F Ø	$\overline{\text{BMB 6}}$	IOT				B T 1				6
				D F 1	$\overline{\text{BMB 7}}$	OPR				POWER CLEAR		SC 1		7
				D F 2	$\overline{\text{BMB 8}}$	FETCH						SC 2		8
				IF Ø	PAUSE	EXEC.						SC 3		9
				IF 1	I O N	DEFER						SC 4		10
MA 11	PC 11	MB 11	AC 11	IF 2	R U N	BRK.	BIT 11	BIT 11	BIT 11	POWER O K	BMB 11	SC 5	BAC 11	11

Figure 3—Group indicator table

```
/APCS-6L  - TAPE 1
/TESTS 1-17 ARE 1 CYCLE INST.
/
/CLEAR 8/I  (SP-ST,SS)
/INITIALIZE TESTER
/LOAD FETCH REGISTER (SEE FR FOR NUMBER)
/LOAD MARGIN REGISTER (OPEN ENDED TIMING)
/LOAD ADDRESS, EXAMINE
/START SW
/
/LOAD MARGIN REG
/PULSE BY PULSE MODE
/
0001 L  NO MEM DONE          /FROM EXAMINE SW. MEMORY CYCLE
0001 LE MA  G2576
0001 LE PC  G2577
0001 LE MB  G0000
0001 LE AC  G0000
0001 LE GA  G1400
0001 LE GB  G0000
0001 LE MS  G4000
0001 LE FR  G5325
0001 LE DR  G0000
0001 LE ER  G0000
0001 LE TR  G0141
0001 LE BI  G0000
0001 LE SC  G0000
0001 LE BC  G0000

0001 F  NO MEM DONE
0001 F4 MA  G2525
0001 F4 PC  G2526
0001 F4 MB  G5325
0001 F4 AC  G0000
0001 F4 GA  G1400
0001 F4 GB  G0321
0001 F4 MS  G0110
0001 F4 FR  G5325
0001 F4 DR  G0000
0001 F4 ER  G0000
0001 F4 TR  G0161
0001 F4 BI  G5325
```

Figure 4—Error message examples

is the most important phase of any attempt at solving a problem.

Error format

- Column 1 designates the Test Number within the program.
- Column 2 is the check point number within a test. It may be a number (1 to 7) or a letter-number combination (i.e., F3, D3, E1, etc.). If a letter and number, the letter is the cycle name (i.e., F = fetch, D = Defer, etc.) and the number is the time state in which the test is performed. If a single number, the cycle and time should be given under the particular tests format.
- No Mem Done—This indicates the memory was cycled by either a switch function or by programming command and did not complete its cycle.
- Column 3 gives the abbreviation of the register in which an error occurred. (See master chart.) Numbers loaded into the 8/I will appear in some of these registers in the printout marked "G" (Good) after the register. Individual test formats will designate when this is true.
- Column 4 is the "G" or Good number. That is the number or instruction loaded into or predicted from the 8/I under test. They are printed in octal.
- Column 5 is the "B" or Bad number. These are not included in these listings. The "B" column gives the incorrect number from the 8/I. They are printed in octal.

Phase 2

The System Controller is composed of a PDP-8, I/O peripheral equipment, an Inter-Processor Buffer and a residing software package.

The objective of Phase 2 is to provide the technician with a quick means of reloading a Maintenance Program to worst case Margin Test a PDP-8/I. It is also used to automatically accept unattended computers.

The System Controller can be used as a high-speed program loader, or to automatically run a computer under test with diagnostic programs in a predetermined sequence. The programs are stored on a disk at the System Controller. Many stations can be tied to the controller's bus for access to the diagnostic programs. The number of stations is limited only by the dwell time for servicing a station.

Drawn in Figure 5 is a block diagram of the system.

Each station has a console which has switches that the operator uses to select a diagnostic program. He then presses a switch to indicate to the System Controller that he wants a program. Indicators are used

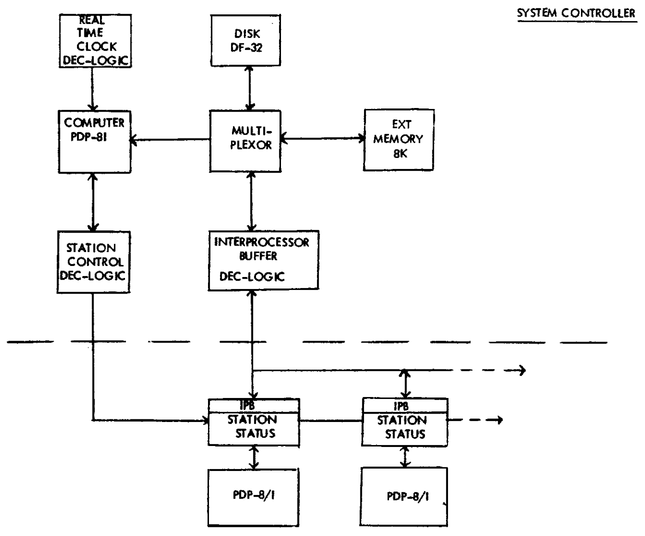


Figure 5—Flow chart of Phase 2 test system

to inform the operator of a successful load or an error condition.

The System Controller connects one station at a time to the bus by enabling that station's gates. The station is then interrogated for program selection. If a program is requested, the desired program is brought from the source disk to the extended memory field of the controller and checksummed. From the extended memory, the program is then loaded into the computer under test with the aid of an Inter-Processor Buffer (computer to computer transfer). After the program has been loaded, the extended memory of the controller is cleared and the program is then sent back to it from the computer under test. The controller then makes a checksum test to determine if the program was successfully loaded If a bit were picked up or lost during the transfer, an error condition would exist. The operator is notified. via the station indicators whether there was an error or successful load (Program loading with overhead 2 seconds).

When being used as a high-speed loader (Figure 5/P), the operator manually calls for the individual Processor Tests from the System Controller. These tests are designed to catch worst case failures caused by temperature, vibration and voltage margins. These are our standard field-oriented maintenance programs.

When being used in the auto accept mode (Figure 5/PA) following a successful load, the computer under test will be told to jump to the beginning of the program and run it for some specified length of time. While the computer under test is running, the controller monitors each on-line station for errors. If an error exists the Sys-

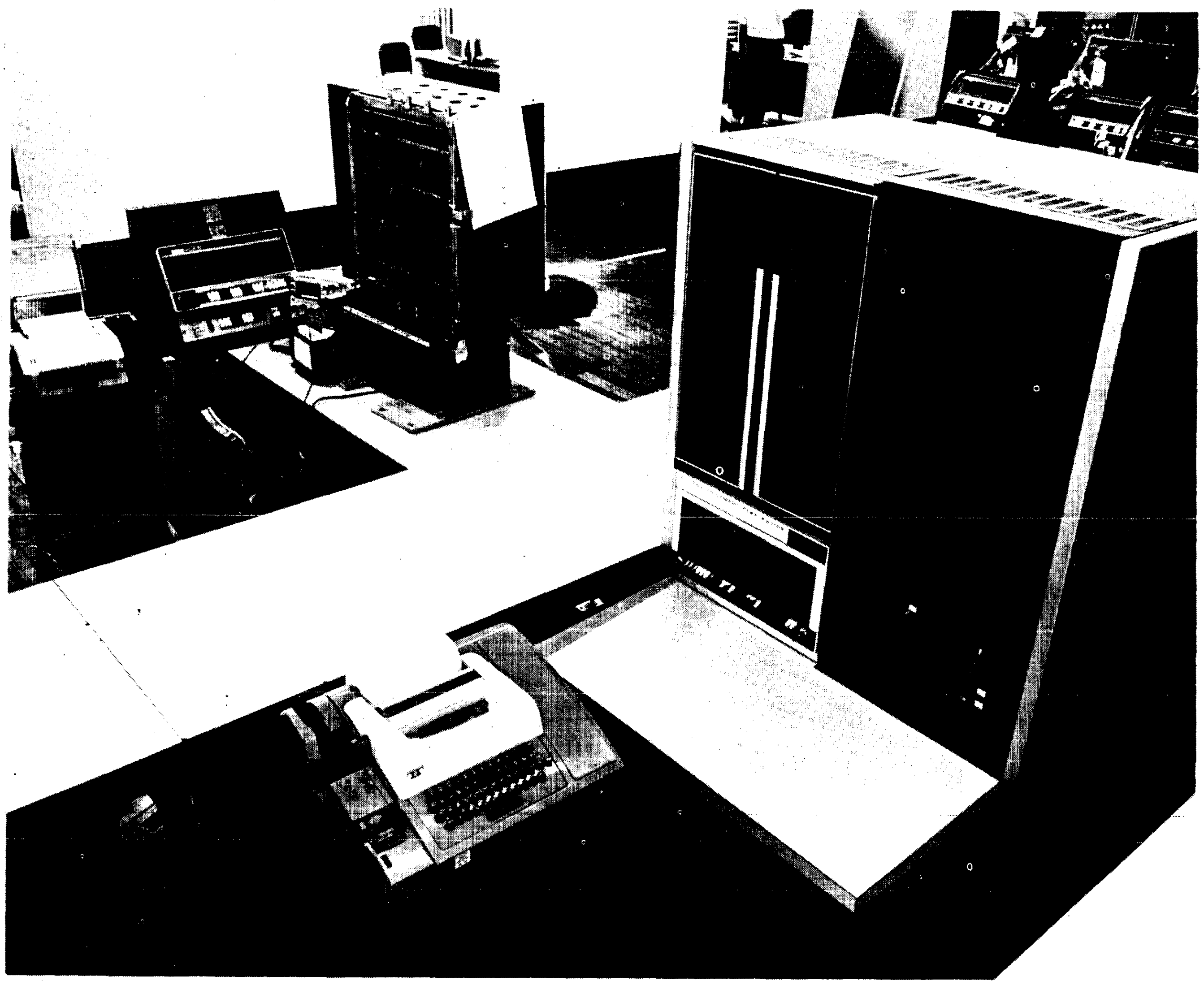


Figure 5/P—Actual Phase 2 environmental test system

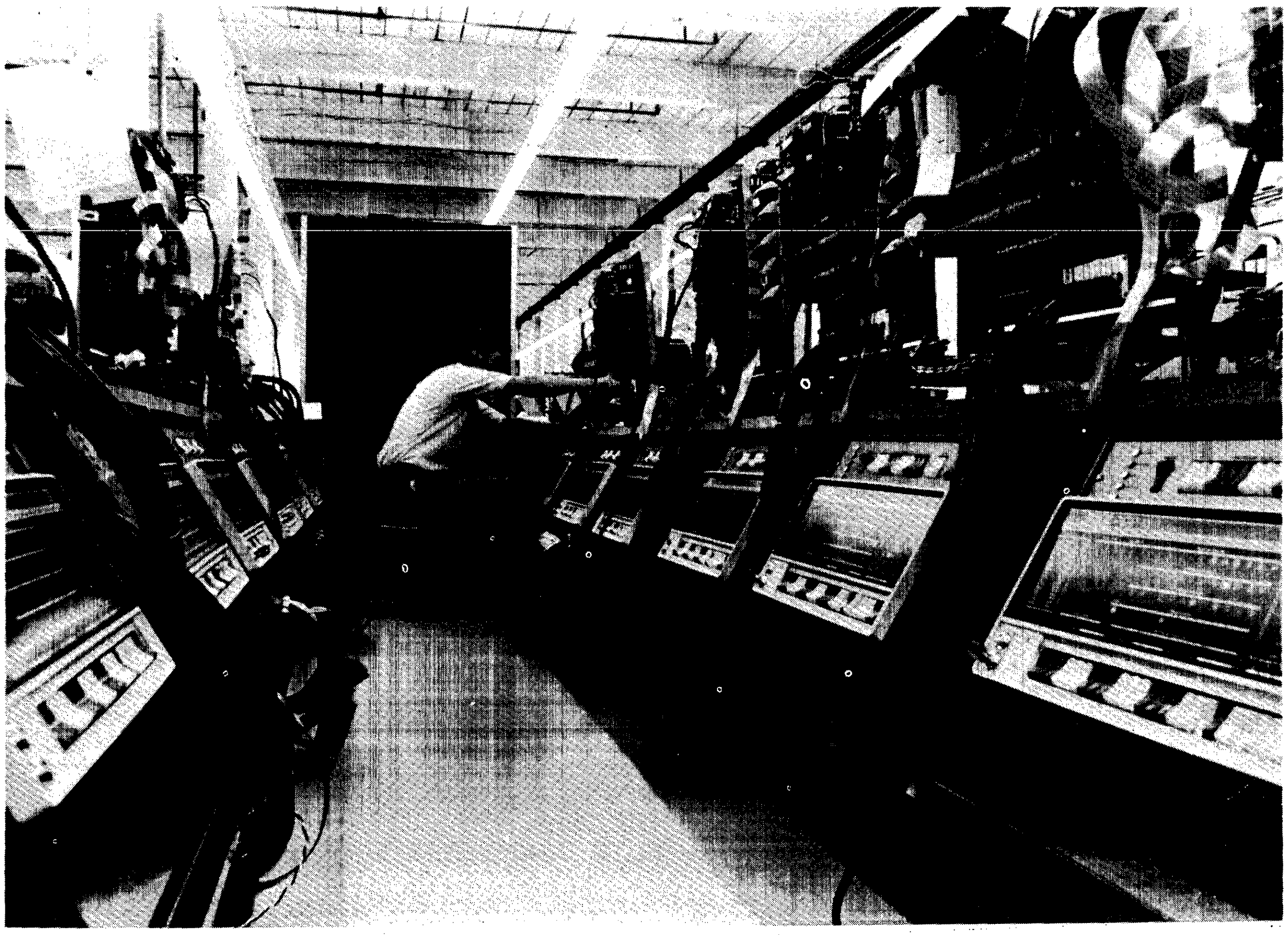


Figure 5/PA—Actual Phase 2 auto-acceptance test system

tem Controller will output the error information on a printer as well as the operators console. The controller has the ability to re-try running that program to gather further information as to whether is is a transient or a repetitive failure. If the error is catastrophic, the station will be disconnected from further service. If no errors have developed after a specified length of time, the next diagnostic program is loaded and run. After all diagnostic programs have been run successfully, the controller will type an accept message for that station and start the routine over again. After a specified number of hours of error free operation, the computer under test is ready to be shipped. The auto accept mode of operation does not require any operator intervention.

CONCLUSIONS

The proposed system provides a very effective tool for use in debugging new computer central processors at a production facility. Any malfunction that may exist within most computer control processors can be detected and found by using this system. In general these problems will be found easier when using the test system than when running a diagnostic program in a free standing machine.

While many of the details of this system are oriented towards computers produced by Digital Equipment Corporation, I believe that the basic idea can be adapted to a wide variety of computers, produced by many manufacturers.

The advantages of using a computer test system for checkout on a computer production line are fast turn on time, less experienced test technicians, resulting in greater product reliability at lower cost.

Cryptographic techniques for computers

by DENNIE VAN TASSEL

San Jose State College
San Jose, California

INTRODUCTION

The use of systems of secret communications as an economical method to increase the security of confidential computerized files has stimulated much interest. Just recently during Congressman Gallagher's Congressional hearings on privacy, it was repeatedly suggested that cryptographic-type protection should be used for data communication lines and storage of confidential information in order to make eavesdropping an extremely difficult task.[1] Today, one finds the very nature of computerized information systems actually facilitates its unlawful reproduction and transmission to anyone with the tools and know-how. Unlike information which is stored with scrambling techniques, information stored in clear form requires no sophisticated technology, nor complex deciphering systems for either decoding or dissemination.[2] More importantly, there is good reason to assume that organized crime and industrial spies have, or will have, the knowledge and the financial resources necessary to acquire and misuse the information in most systems now being considered, including the tapping of communication lines. Finally, once a piece of information is lost, its original confidentiality can never be regained. Since information which has previously been scattered among several rather protected and widespread sources is being collected into one place, wholesale theft of information is very likely to become a continuing fact of life for the American public.

This new pattern of computer misuse must be discouraged by the imposition of severe sanctions as well as clearly defined safeguards. One dangerous factor of keeping information confidential is that our society does not believe its own pronouncements about the right of secrecy or privacy. In general, the majority of people will support its own claim on the right of keeping some information confidential but at the same time will not demand prohibitive action against both governmental and private snooping. While it is obvious that this reaction says little for society's capacity to be honest and rational, equally important, this trend strengthens the possibility that individual privacy may vanish like the American buffalo.

The main focus of this article is on cryptographic techniques. This leads to something of a distortion. While cryptographic techniques are an essential means of keeping information confidential they are a symptom rather than a cure of the larger problem of privacy. While this is not the appropriate place to expound the question of the right of privacy and the relationship of privacy and the computer, it has been necessary to give the problem minimal consideration here.

A high degree of secrecy at minimum cost can be achieved through the use of cryptographic techniques for the protection of confidential information. The general principle behind the use of cryptographic techniques to protect confidential information is that if unauthorized use is inexpedient and costly the price of such use is raised to such an extent that it is generally uneconomical to attempt it. The same principle is used in building bank vaults.

In order to explain the use of cryptographic techniques for computers it will be necessary to preface the discussion with a brief definition of terms. A PLAINTEXT message is the input message. A CIPHER or CRYPTOGRAM is the output message; that is, the message after it has been changed to hide its meaning. CRYPTANALYSIS is the act of resolving cryptograms into their intelligible texts without having possession of the system or key employed. Throughout this paper the word ENEMY will be used to designate any persons *not* authorized access to the messages.

There are two principal classes of cryptography, transposition and substitution. A TRANSPOSITION cipher is one in which the letters of the plaintext are unchanged, but its order is rearranged so that the cipher message apparently conceals the clear message. A MONOLITERAL transposition is one in which the transposed elements are the single letters of the plaintext. If the transposition operates on groups of letters or words, the transposition is called POLYLITERAL. The agreed upon manner of writing the clear message into the agreed upon pattern is called the INSCRIPTION of the message. The method of taking off the letters from the cipher sequence is called the TRANSCRIPTION.

In a SUBSTITUTION cipher the elements of the plaintext keep their relative position, but they are replaced in the cipher text by other letters or symbols. Substitution symbols are MONOALPHABETIC if a single cipher alphabet is used and POLYALPHABETIC if two or more cipher alphabets are used. When the substitution involves one letter at a time the cipher is MONOGRAPHIC. In a POLYGRAPHIC system the substitution is by pairs of letters or larger groups. Most cipher systems consist of two parts; a general method which is fixed, and a key-variable which changes at the will of the correspondents and which controls the operation of the basic method. Keys are either literal, consisting of words, phrases or sentences; or numerical composed of figures.[3] Examples of keys will be provided with each method.

Transposition methods

Transposition techniques consist of changing the natural order of a record so that the original meaning is hidden. A common game with children is to write a message backwards. The record VAN TASSEL would be enciphered LESSAT NAV. Another simple example is when adjoining characters are switched; using the same record, VAN TASSEL becomes AV NATSSLE. Another method of transposing is to take all the even numbered characters within a record and move them to the first half of the record, and move the odd numbered characters to the last half of the record, as follows: VAN TASSEL becomes A ASLVNTSE. These are simple examples of transposition methods.

During the civil war a simple transposition called the RAIL FENCE transposition was commonly used. In this system a record containing a Social Security Number 503-40-1687 and an hourly rate of $3.12, (5034016870312), is inscribed along a rail fence as follows:

```
5 3 0 6 7 3 2
 0 4 1 8 0 1
```

Then the transcription is taken off row by row in groups of five as shown below:

```
53067  32041  801
```

The rail fence method can be modified to a 3-row, 4-row, or n-row; however the security obtained is minimal. The key in this method is the number of rows used.

There are other ways in which records can be scrambled. One of the most common makes use of rectangles to scramble the message. We can inscribe the alphabet into a rectangle by using a vertical path as follows:

```
A  E  I  M  Q  U  Y
B  F  J  N  R  V  Z
C  G  K  O  S  W
D  H  L  P  T  X
```

This transcription consists of taking the elements off horizontally, as follows:

AEIMQ UYBFJ NRVZC GKOSW DHLPT X.

There are many variations of the possible inscription routes and as many for the transcription. In fact, if the record is n-elements long there are n! (n factorial) possible orderings of the elements. Using conventional geometric inscriptions we may obtain the following routes:

1 Horizontal
2. Vertical
3. Spiral
4. Diagonal

A horizontal route can have all the rows inscribed in the same direction or in alternating directions. The same is true for the vertical route. A spiral route may be clockwise or counter-clockwise. And a diagonal route may be diagonal upwards throughout or diagonal downwards throughout, or it may begin either upward or downward and alternate its direction while maintaining a diagonal path. Thus, there are ten routes in inscription beginning with each corner of the rectangle, i.e., 2 horizontal, 2 vertical, 2 spiral, and 4 diagonal routes. Therefore, there are 40 different possible paths using conventional routes. This method is particularly fitted to computer usage because of the ease of setting up matrixes and modifying the path of transcription.

The size of the rectangle has much to do with the amount of security obtained. If a rectangle of 64 elements is used there are five possible rectangles, i.e., 2×32, 4×16, 8×8, 16×4, 32×2. On the other hand when using a rectangle of size 72 elements there are 10 possible rectangles, i.e., 2×36, 3×24, 4×18, 6 × 12, 8 × 9, 9 × 8, 12 × 6, 18 × 4, 24 × 3, 36 × 2. The greater the number of possible rectangles the greater the security.

It is not necessary to use a rectangle that fits the record length exactly. That is, a record of length 26 can be inscribed into a rectangle of 28 elements. The unused cubes are simply ignored while the record is transcribed as it was in the first rectanglar example. Once the size of the rectangule is known, it is much easier for an enemy to decipher the cryptogram; therefore, it is undesirable to use a rectangle the same size as the record. If partially filled rectangles are used, much less information is given on the encipherment method.

Alphabetic keys are often used to encipher a record. Using the key-word LENGTH, the key-word is placed over the columns as follows:

L	E	N	G	T	H
A	B	C	D	E	F
G	H	I	J	K	L
M	N	O	P	Q	R
S	T	U	V	W	X

Next, the letters in the key-word are put in alphabetic order and the respective columns are shifted along with the letters of the key-word as follows:

E	G	H	L	N	T
B	D	F	A	C	E
H	J	L	G	I	K
N	P	R	M	O	Q
T	V	X	S	U	W

Quite often a key-word is also used on the rows in the same manner. This method of transposition is called NIHILIST, named after the Russian anti-Tsarists who used this method.

Substitution methods

In a substitution cipher the elements of the plaintext keep their relative positions, but they are replaced in the cipher text by other letters or symbols. One of the most simple substitution systems is called a CAESAR system, supposedly used by Julius Caesar. Caesar replaced each plaintext letter by the third letter after the replaced letter in the alphabet. The correspondence of each plaintext letter with its cipher letter is represented in the following table:

plain A B C D E F G H I J K L M N O P

cipher D E F G H I J K L M N O P Q R S
T U V W X Y Z A B C Q R S T U V W X Y Z

The record FORTRAN is enciphered as IRUWUDQ. The process here is similar to module arithmetic with base 26. Sometimes a reversed cipher alphabet is used, but predictably neither method offers a high degree of security.

A key-word method offers a much higher degree of security. The key-word is written first and followed by other letters of the alphabet not in the key-word. Any letter in the key-word which is repeated in the key-word is dropped the second time. If the key-word is CONFIDENTIAL the key-word alphabet would be as follows:

plain A B C D E F G H I J K L M N O P

cipher C O N F I D E T A L B G H J K M
P Q R S U V W X Y Z Q R S T U V W X Y Z

It should be noted that both the letters N and I were repeated in the word CONFIDENTIAL so that when this word was used to set up the key-word alphabet the second N and I were dropped and CONFIDETAL was used. The record FORTRAN is enciphered as DKQSQCJ.

Instead of using a key-word cipher alphabet it is possible to set up a strictly random sequence of the alphabet. But the additional security obtained is not much greater than with the key-word alphabet, and the key-word system lends itself to use by subroutines and simple change of the cipher by merely changing the key-word.

One disadvantage of all the above substitution methods is that once it is set up the cipher never changes. This lessens the security because there are certain statistical and structural characteristics of English words that help break a substitution system of this type. For instance, one is that the most common letter in the alphabet is the letter E, so a cryptanalyst has to merely look for the most common letter and chances are that it is really the letter E. Here in their order of importance is a list of the most common letters of the alphabet E T O A N I R S H. There are also certain structural characteristics in English such as the letter U always follow the letter Q. Another common characteristic in English is that certain digraphs such as TH appear quite often. Addresses, dates, COBOL and FORTRAN programs are especially vulnerable to analysis. By now readers will have observed that the above systems are not invulnerable by any means, and that the art of cryptography must require a more complex system than offered so far.

Important systems to consider are those of polyalphabets, that is, those with more than one alphabet. Clearly, polyalphabets are used to make the enemy's work a little more difficult. A GRONSFELD encipherment uses a numerical key and modifies the traditional Caesar system. Using a key 31206 and the plaintext record PROGRAMMING the following encipherment is obtained;

key	3 1 2 0 6	3 1 2 0 6	3
plain	PROGR	AMMIN	G
cipher	SSQGX	DNOIT	J

To encipher P using the key digit three, simply begin at P and count forward three in the normal alphabet; the substitute is S. To encipher R with key one, begin at R and count forward one in the normal alphabet; the substitute is S. For decipherment, count backward in the alphabet. Two factors are evident: there are 10 possible substitutions here (for the digits 0-9) and we lose some of the weakness of the previous systems.

Giovanni Battista della Porta designed a system in the 16th century that was actually considered completely safe from enemy decipherment for 200 years. Remarkably, the system is probably still safe from anyone but a trained cryptanalyst. Here is the Porta table:

PORTA TABLE

AB	A	B	C	D	E	F	G	H	I	J	K	L	M
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
CD	A	B	C	D	E	F	G	H	I	J	K	L	M
	O	P	Q	R	S	T	U	V	W	X	Y	Z	N
EF	A	B	C	D	E	F	G	H	I	J	K	L	M
	P	Q	R	S	T	U	V	W	X	Y	Z	N	O
GH	A	B	C	D	E	F	G	H	I	J	K	L	M
	Q	R	S	T	U	V	W	X	Y	Z	N	O	P
IJ	A	B	C	D	E	F	G	H	I	J	K	L	M
	R	S	T	U	V	W	X	Y	Z	N	O	P	Q
KL	A	B	C	D	E	F	G	H	I	J	K	L	M
	S	T	U	V	W	X	Y	Z	N	O	P	Q	R
MN	A	B	C	D	E	F	G	H	I	J	K	L	M
	T	U	V	W	X	Y	Z	N	O	P	Q	R	S
OP	A	B	C	D	E	F	G	H	I	J	K	L	M
	U	V	W	X	Y	Z	N	O	P	Q	R	S	T
QR	A	B	C	D	E	F	G	H	I	J	K	L	M
	V	W	X	Y	Z	N	O	P	Q	R	S	T	U
ST	A	B	C	D	E	F	G	H	I	J	K	L	M
	W	X	Y	Z	N	O	P	Q	R	S	T	U	V
UV	A	B	C	D	E	F	G	H	I	J	K	L	M
	X	Y	Z	N	O	P	Q	R	S	T	U	V	W
WX	A	B	C	D	E	F	G	H	I	J	K	L	M
	Y	Z	N	O	P	Q	R	S	T	U	V	W	X
YZ	A	B	C	D	E	F	G	H	I	J	K	L	M
	Z	N	O	P	Q	R	S	T	U	V	W	X	Y

This system uses a key-word and 13 cipher alphabets. If the key-word is HELP and the record to be enciphered is PROGRAMMING the following encipherment is obtained:

key	HELP	HELP	HEL
plain	PROG	RAMM	ING
cipher	MCJN	BPRT	YLY

By referring back to the Porta chart we can see 13 alphabets. In all 13 of these cipher alphabets, the encipherment is reciprocal. In the first alphabet on the chart, for instance, the substitutet for A is N, and the substitute for N is A. This system applies a key-word to determine which alphabet is to be used. If the key-letter is either A or B, the first alphabet is the one to be used.

In the example above the first key-letter is H. Therefore, we go down to the GH alphabet in the Porta table, which is the fourth alphabet from the top. Next we take the plaintext letter under the H in the example, which is P, and find the letter M above the P in GH Porta alphabet. M is the cipher letter corresponding to P. This system is an improvement on the Gronsfeld system because of the greater number of alphabets.

We now come to a system developed by Sir Francis Beaufort which has of course the name 'the Beaufort method.' Not unlike the Porta system the Beaufort system also makes use of a key-word and uses a Beaufort Table (shown below).

BEAUFORT TABLE

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A

In this table we have 27 alphabets in which all four of the outside alphabets are exactly alike. Using the same key-word HELP and the plaintext record PROGRAMMING we will get the following encipherment:

key	HELP	HELP	HEL
plain	PROG	RAMM	ING
cipher	INDR	KWBX	BJV

To encipher by this method we start with the key-letter H (at any border) and trace inward to the plaintext letter P, turn sideways and emerge at the border letter I which is the encipherment of P using the key-letter H.

While these substitution systems have been set up for a 26-element character set, they can all be expanded to any desirable character set such as a 36-element character set which would include the 26 elements of the alphabet and the numbers 0-9 or perhaps even one of the 48- or 60-element character sets commonly used in computers.

Advanced cryptographic techniques

The main focus of this paper has been an introductory look at many general methods of cryptography. Indeed, while each slight variation can add or lessen the security acquired from the individual system the many possible modifications of each system have not been included because of the limited scope of this paper.

Yet it is necessary at this point to raise some of the more advanced cryptographic techniques. Quite often substitution and transposition methods are combined. First, a record is put through a transposition and next a substitution is applied. This will surely increase the security even if both the transposition and substitution are quite simple.

A double transposition or double substitution can be used and although it doesn't lower the security neither does it always significantly raise the security. An example of a double substitution that would actually reduce the security is to use two Caesar substitutions that were actually inverses of each other. While this may seem ridiculous it is possible to use two substitutions or transpositions without noticing that they are actually partial inverses.

Valuable information to possess when tapping a communication line is to know exactly when information is being transmitted. To avoid releasing even this small amount of information, strings of random digits are often transmitted when the system is not being used.[4] This necessarily puts the intercepter in the position of first having to determine which string of digits is useful information and which is garbage information; otherwise he will spend days trying to decipher something that doesn't mean anything in the first place. The same principle can be used with stored memory files. Unused memory space is inserted with strings of random digits.

Messages are also split up into segments so that part of the message is sent at one time and the rest is sent at another time, possibly even by a different system of communications. The same principle can be used on stored memory files so that records are divided up into segments and spread through two or more storage devices so decipherment could only begin after the different sources were properly brought together.

Another technique is to add a generated random binary sequence to records as they are being stored in memory. It is possible to add a different random number sequence to each record so that decipherment of one record by an enemy will give no information on how to decipher the next record. This can be a rather discouraging prospect for even the most persistent cryptanalyst with a computer.

Indeed, the list of cyptographic techniques is as endless as the inventive genius of man. All one has to do is transmit the cryptographic key in some non-interceptable way and to have all operations reversible (nonsingular). As to what constitutes the perfect cipher, Francis Bacon says: "that they be not laborious to write and read; that they be impossible to decipher; and in some cases, that they be without suspicion."

REFERENCES

1 *The computer and invasion of privacy*
Hearings Before a Subcommittee on Government Operations House of Representatives July 26–28 1966
U.S. Government Printing Office Washington D.C. 1966

2 H E PETERSON R TURN
System implication of information privacy
Proc S J C C 1967

3 D D MILLIKIN
Elementary crytography and crytanalysis
New York University New York 1943

4 P BARAN
On distributed communications: IX. security, secrecy and tamper-free considerations
RM–3765–PR The RAND Corporation Santa Monica California August 1964

Montessori techniques applied to programmer training in a workshop environment

by ELIZABETH R. ALEXANDER

California Division of Highways
Sacramento, California

INTRODUCTION

This paper describes a unique workshop structure based on the Montessori Method and utilizing both vertical and horizontal interaction in the training of systems programmers.

The scope of this paper covers the definition of the training need, the objectives of the workshop, the training curriculum, the selection process, the application of certain Montessori techniques in the training of programmers, the measurements for evaluating success, the results of the workshop to date, and the *planned*, continued upward development of programmers.

Training need

The California State Division of Highways' unsatisfactory experience with on-the-job and vendor training resulted in our decision to develop and implement our own in-house programmer training workshop. Prior to August, 1966, we depended upon on-the-job training supplemented by vendor courses.

On-the-job training was constantly interrupted by production schedules; production supervisors did not allow adequate time for training, and they were not necessarily qualified or motivated to train.

Vendor training objectives tended toward course completion and coverage of material. The courses were generalized, scheduled at inconvenient times and did not cover the total subject matter necessary to the training of a thoroughly productive trainee systems programmer.

Individual self-discipline, motivation and ambition for increased levels of responsibility did not result from these approaches toward training.

In July, 1966, we developed a curriculum for an in-house systems programmer training workshop, and applied certain training methods formulated by Maria Montessori.[1] These methods are the right of the student to be active, to explore his environment and to develop inner resources through investigation and creative effort. The environment should be such as to give scope to the individual's inner resources, to direct these resources and most of all to call them forth. The instructor's task is one of assisting, watching, encouraging and inducing rather than interfering, prescribing or restricting.

To date, seventy-two systems programmer *trainees* have graduated from eight classes where these techniques have been applied.

A systems programmer *trainee* within our installation is one whose basic training includes concepts considerably beyond the fundamentals of applications programming. A graduate from our systems program, mer trainee workshop must also have a broad, basic understanding of the logic of the operating system-data management, system library functions, catalogue procedures, the sophistications and efficiencies of Job Control Language, the purpose of system control blocks, the various access methods, available utilities, and core dump analysis.

Graduates from our workshop are eligible for consideration for rotation into systems software programming after a minimum of six month's experience in applications programming. Upon rotation into the systems software programming unit, they receive additional training in operating system coding, teleprocessing access methods, system generation and details concerning the logic of the operating system under a multiple variable task environment.

The objectives of the workshop

The objectives of our programmer training workshop are:

1. The development of competent trainee *systems* programmers who can take immediate responsibility for complex assignments.
2. The development of individual self-discipline, professional motivation and endeavor towards reaching levels of increased responsibility such that it will continue after graduation from the workshop.
3. The assurance of a steady flow of capable personnel, qualified and trained, to fill programming positions which occur through promotions and attrition. We cannot afford to be dependent upon the indispensable programmer.
4. The programmer training workshop is only a first step in the system programmer's development; it serves to reduce attrition in the long run by providing opportunities limited only by the individual's interest and ability.

The training curriculum

The training curriculum is dynamic in nature. It is subject to revision based on our current hardware/software environment and advancements in the state-of-the-arts which have practical application within our installation.

At the present time our environment consists of a System/360 Model 65 with one million bytes of core storage, twelve tape drives, two 2314 disk storage devices, one data cell with another to be delivered in August, two high-speed printers, a drum, a Cal-Comp plotter, remote job entry with 2780's in eleven districts, six 2260 display units in the software and programming units, with additional 2260's on order for users.

We are under a full operating system with the Faster generalized file processor and are in a multiple variable task environment.

The curriculum developed for the workshop is divided into two parts. The first section covers programming concepts and the hardware and software of the system and comprises eight weeks. The remaining four weeks are devoted to a real, live production environment within the workshop in which each trainee is given a fairly complex production program to chart, code, compile and test, and which will actually be used by the installation. Each trainee is given a completely different program.

Implementation of the training workshop

Selection process

Candidates for our training workshop must pass a State version of the IBM Programmer Aptitude Test. The cut-off point is 70 percent. This test eliminates 50 percent of the candidate group. Successful candidates are further evaluated through in-depth interviews.

Questions are asked during the interview which will expose the candidate's self-image and professional goals. The candidate is asked to relate experiences which will substantiate the salient points of the self-image. The self-image and goals are then evaluated against the following behavioral criteria:

1. A strongly developed sense of personal motivation toward a long-term career in data processing.
2. A basic character that is conducive to developing a high degree of self-discipline.
3. An enthusiastic interest in working at a detailed level for long periods without frustration.
4. The ability to work under pressure to meet critical deadlines.
5. The ability to communicate.

Enrollment in our training workshop is limited to twelve trainees. This is the maximum number of students which we feel will allow us to maintain the desired flexibility of working conditions.

Unique Montessori environment

Our training environment is unique because it applies Montessori Methods to the training of systems programmers. These methods are the development of self-discipline through freedom, an environment which challenges and motivates on an individual basis and encourages individual growth toward full potential.

The workshop is structured to appear to be a free form in-group atmosphere. It is, in fact, a carefully controlled and disciplined environment where the ground rules for expected performance within the workshop emphasize the professional attitude expected in the production environment of the installation. The workshop stresses group interaction and individual self-discipline. It utilizes the unusual horizontal interaction of students training each other as well as the traditional vertical interaction of the instructor training the students.

Ground rules for expected performance

Ground rules for expected performance are outlined for the trainees.

It is emphasized that each trainee has been carefully selected for this workshop.

The workshop has been implemented to produce qualified, first level systems programmer trainees. It was not implemented to produce coders of higher level languages. It is expected that the trainees will keep this objective before them as they proceed through a difficult curriculum.

The curriculum requires an average of three hours of voluntary homework assignment each day. These homework assignments make it possible to cover an extensive amount of academic material within an eight week period, evaluate the professional motivation of the trainee, and apply pressures of motivation and self-discipline.

This workshop does not graduate a trainee who demonstrates a lack of ability or who fails to meet the expected high level of performance. The trainees are told that the expected attrition rate within the workshop is 10 to 20 percent and that there will be no exceptions to the standards of expected performance.

The graduates from this workshop are expected to be able to immediately assume responsibility for complex programming assignments with minimal supervision.

Application of the Montessori method

The application of the Montessori Method to our training workshop has been achieved in the following areas:

1. *Intense self-discipline through freedom*

   The trainee has been encouraged to discipline his own study habits and not be concerned with the pace at which other trainees may learn. Observation has shown that when a trainee attempts to emulate the study habits of a peer, he frustrates his natural mode of self-discipline. This results in reduced performance. When the trainee returns to his own specific form of self-discipline, his level of performance rises again.

   The trainee is required to demonstrate responsible initiative in coming to the training instructor for individual guidance and instruction. Every opportunity is given for special tutoring at a time acceptable to his schedule. He must, however, demonstrate a desire for help in achieving his goal. If the instructor wishes to remind a particular student that special tutoring is available, the reminder is directed toward the entire group; not toward the individual. This approach forces the student to reconsider the advantages of accepting help and the alternative of possible failure to pass successfully through the curriculum. It emphasizes to him the need for self-discipline through freedom of action on his part in taking the proper initiative. It is made clear to the trainee that he is responsible for properly interpreting and understanding the subject matter. Quizzes and examinations are open book to discourage memorization.

   The trainee is allowed considerable freedom in his individual ability to demonstrate self-discipline. A prime example of this freedom occurs during the study periods. The trainee is not actively monitored during these periods. He has the responsibility to be prepared to participate in the next scheduled group discussion. He cannot be a disruptive force to others who are studying. He is given the freedom to study by himself, study in a small group cluster where there is a horizontal exchange of ideas and a process of teaching each other, or he cannot study at all.

   During the study periods, the instructor will sometimes leave the environment for a half-hour or so. This is done to encourage a totally free environment for individual self-discipline on an active basis.

   The training instructor keeps a detailed, daily diary on each trainee. Particular attention is given to recording individual critical incidents. This diary serves as a reminder to the instructor during the weekly evaluation and counseling session with the student. It also is used to substantiate an outstanding performance report or as detailed documentation required for recommending rejection from the workshop.

2. *Challenging and motivating the individual*

   The trainee is given increasingly difficult assignments. He is told that the curriculum covers in-depth systems programming concepts. On the first day of the workshop he is given some thirty-five technical manuals ranging from basic programming through details concerning the internal logic of the operating system. He is told that he will be expected to understand and apply the technical subject matter covered within these manuals and that he has eight weeks to reach these expectations. The intense impact of receiving all thirty-five manuals[2] at one time has the effect of challenging and motivating the trainee toward full achievement of these goals.

   The trainee is told that upon leaving the workshop environment he will be expected to be immediately productive and contribute to the upgrading of the installation. His training has prepared him to move in several directions of individual development. Within a relatively short period of six to nine months he will be expected to become a responsible programmer of large Management Information Systems applications, a fully qualified software programmer responsible for the operating system or a technical specialist in teleprocessing, generalized file processors such as Gentry or Mark IV, Graphics or any other advancements in the state-of-the-arts that the installation implements.

The trainee is encouraged to challenge his environment at any point. He is asked to become an experimentalist—to not be afraid to try a technical innovation which has not yet been implemented within the installation but which seems to be a practical approach and is proven to be theoretically possible. He is encouraged to be an innovationalist—to not be inhibited in making suggestions for improvement. The philosophy of the workshop is that the newest employee can possibly make a major contribution to progress within the installation.

3. *Special emphasis on individuality*

The workshop encourages the trainee to have a deep curiosity for exploring beyond the subject matter covered within the curriculum. The trainee is encouraged by his own motivation to do individual assignments beyond those required by the curriculum and to report back to the group. Examples of such individual assignments have been the internal logic manuals on the Compiler, Linkage Editor and Input/Output Control System. The IBM Systems Journal, Volume Three, Numbers Two and Three, 1964, with A Formal Description of the System/360 by Dr. Kenneth Iverson,[3] has been given as a special reading assignment to several trainees who demonstrated interest and the ability to absorb the material. One trainee, especially qualified by his background, was encouraged to give a session to the group on vector analysis.

This initial interest on the part of the trainee has usually resulted in identifying a direction for continued self-development which has been beneficial to himself and to the installation.

Horizontal and vertical interactions

The physical environment of the workshop is made conducive to horizontal interaction in the exchange of ideas among the trainees. The desks are placed in a conference arrangement so that the students face each other. One desk in the group is reserved for the instructor, who, while sitting at this desk, enters the group involvement on an active basis and in fact becomes a member of the group.

The discussion periods are a complex environment utilizing the Montessori Methods of individual self-discipline, motivation and endeavor. Superimposed is a group dynamic structure which is best emphasized by the fact that a cooperative rather than competitive attitude is developed among the group. Group motivation is stressed.

At times the group actively takes over and runs its own training. It is they who select a peer to go up to the blackboard and summarize the key points of their discussion; and it is they who monitor their own horizontal interaction. At such times the instructor remains outside the group and does not become actively involved unless the group loses control over the discussion.

At other times the vertical interaction is evident between the instructor and individual students or between the instructor and the group as a whole. The instructor may have an open discussion with one member of the group. He may by his singular attitude indicate to the rest of the class that he does not want any group interference in this vertical interaction with a selected member from the group.

The group may "go critical" with respect to vertical interaction with the instructor. There has been in every workshop so far a crisis point about half way through the curriculum where the group attempts to resist learning any more, applying any further self-discipline and individual motivation. This critical period is usually of short duration, about a day, and marks the point at which the group learning can go in two directions. The whole situation can fall on its own weight and from then on only desultory learning and results can be expected. Or, the environment-cum-people combination can go "critical" and a kind of self-sustaining *group* reaction is initiated which releases enough motivational and learning energy to carry the group through the increasing but controlled pressures of the workshop. This occurs primarily because of the unifying force of the instructor's own intense enthusiasm and motivation. In all eight workshops so far the students have reacted with greater self-discipline to the challenge of completing the course and becoming productive systems programmer trainees.

Measurements for evaluating success

The following measurements have been applied in evaluating the success of our workshop:

Measurements within the workshop environment

1. Observation of individual motivation and self-discipline in the mastery of the subject matter. These observations are recorded in the daily diary of critical incidents and are periodically evaluated by the training instructor.
2. Successful development to the level of systems programmer trainee.
3. Performance in written quizzes, examinations and case studies.
4. Evaluation of the trainee's ability to relate academic subject matter to the first production assignment.

Long term measurements

The long term measurements of the individual are:

1. A continued cooperative attitude among the peers working in a production unit.
2. Continued motivation toward self-development. Each employee has an individual Employee Development Plan which is kept current by his production supervisor.
3. The ability to handle complex systems assignments in a timely and efficient manner and to meet critical deadlines under pressure with few compilations and tests.
4. Self-discipline which results in minimal supervision and is demonstrated by adherence to standards and guidelines of the installation. This measurement can best be taken in terms of adherence to the requirement of complete documentation of a given system.
5. The continued advancement to ever increasing areas of responsibility within the installation and success in passing examinations for higher work classifications.
6. The periodic, written evaluation of the individual's performance by the production supervisor.

Results of our training workshop

The results of our training workshop for *systems* programmers has been twofold. There has been an immediate upgrading of programming performance and we have identified a strong need for individual development and continued training for all 209 persons employed within our installation.

The upgrading of programming performance by graduated trainees has been demonstrated in an outstanding manner.

The trainees have continued to demonstrate a high motivation toward long term professional careers. This motivation is partly due to basic personality characteristics which were carefully selected during the in-depth interview and to the fact that the training workshop strengthened this natural motivation.

The supervisors have observed that the expected level of accuracy and self-discipline is being maintained. These programmers require minimal supervision.

Sophisticated production assignments normally given to experienced programmers have been given to the *system* programmer trainees soon after graduation from the workshop. These are often large scale, complex programs involving 50 to 100K core storage, intricate relational editing techniques, table searches, complex mathematical calculations and dense reporting formats. Our applications involve the construction of large information systems within the areas of traffic control, urban planning, fiscal management, administrative support and engineering project control as well as complex programming in the areas of bridge, vertical alignment, traverse and earthwork. Graduates from our trainee workshop are expected to become immediately productive in these areas. Those who rotate into software programming have responsibility for the operating system, the writing of all in-house utilities and subroutines as well as being responsible for maintaining an in-depth understanding of remote job entry and teleprocessing capabilities and the software capabilities of any large generalized file processors currently in use.

The programs written by these former trainees have been logically planned using truth tables and modular design and are fully documented. The number of compilations and tests has been greatly reduced.

These graduates from the training workshop have consistently demonstrated competence and understanding of the complex capabilities of the system. They have been articulate in expressing knowledge of the system and they have demonstrated ability to explore and implement those areas of software which result in the saving of machine time.

A typical example of such a contribution can be illustrated in the fact that a graduate with less than six months of production programming experience researched the feasibility, recommended and then imple-
researched the feasibility, recommend and then implemented the conversion of a large sequential data base to indexed sequential access at a time when *ISAM* was a relatively new and untried quantity.

These systems programmers have demonstrated ability to maintain and modify extremely large and complex systems and the ability to consistently produce under critical deadlines.

Their continued interest in the field of data processing has been evidenced by their active participation in professional associations and their enthusiastic attendance at seminars and symposiums which have covered advancements in the state-of-the-arts. They have continued to demonstrate highly developed study habits which have resulted in their being thoroughly familiar with the subject matter presented in new manuals and publications. All of the former trainees have clearly defined goals for their own long term professional development. Of the seventy-two systems programmers who have graduated from the training workshop, approximately two thirds have returned to take evening courses at college toward a higher degree.

Four new programming units have been created entirely from graduates of the training workshop. These production units have maintained highly satisfactory levels of programming performance.

Ten programmers, who received their training in the workshop, have become part-time instructors in advanced techniques sessions and workshops for programmers.

Ten of the graduates from the training workshop are now highly qualified software specialists responsible for all aspects of maintaining our operating system and implementing teleprocessing, partitioning under multiprogramming, *Faster* and the system utilities.

The lead *systems* programmers in every production unit have graduated from our training workshop within the last year and a half. Two graduates have underfilled temporarily as supervisors of programming units.

Attrition has been noticeably reduced since the implementation of the training workshop from 27 to 14 percent in the first year after implementation of the first workshop. At the present time our attrition is so reduced that we do not anticipate the need for an entry level systems programmer trainee workshop before July, 1969.

The areas of continued training and self-development have been identified in the following ways:

1. The supervisors and experienced programmers have requested continued training for themselves and these training programs are currently being developed for them.
2. We also have an individualized employee development plan[4] which includes every member of our staff and involves 209 employees in 41 computer systems classifications. The basis of this plan is: (a) A determination of skills and knowledges required and desired for each and every position and (b) each employee's background of education and experience. The differences between these two elements becomes a base for our individualized development plan.

   This is not a "one-shot" or lightly considered plan. The dynamic nature of both computer technology and information usage necessitates continuing review, updating, and planning. In effect this plan constitutes a contract with each employee outlining management responsibility and the employee's responsibility for his own self-development. The plan may include anything from college courses to management development, depending upon a mutual agreement as to need.
3. An up-to-date library of current periodicals, journals, and books is being instituted by our management.
4. The need for a full-time training program for Computer Systems has been identified. We have developed and implemented a training workshop for systems analysts utilizing the same involutional group dynamic structure training techniques. To date, four such systems analyst training workshops have been implemented since September, 1968. We are currently developing several workshops for programmers with 12-18 months and 18-24 months experience as well as a series of workshops for computer operators.

CONCLUSION

Our efforts in systems programmer training have had immediate and long-range results. We have achieved the level and quality of programming skills necessary for successful operation in our third generation environment. We have recognized the need for both initial and continued training.

We have recognized immediate results from the application of the Montessori Methods emphasizing individual self-discipline through freedom, individual motivation and endeavor towards reaching levels of increased responsibility. Our present and future workshops will continue to utilize these training techniques which are emphasized by the Montessori Method of teaching.

A formal individual development for each of our 209 employees has been implemented. Most important, management has recognized that the development of personnel on a planned, continuous basis is the key to successful achievement of our long-range operating goals.

REFERENCES

1 M MONTESSORI
The Montessori method
Second Edition Frederick A Stokes Co New York 1912

2 E R ALEXANDER
Third generation programmer training—the workshop approach
Proc Fifth Annual Computer Personnel Research Conference 1967

3 A D FALKOFF K E IVERSON E H SUSSENGUTH
A formal description of systems/360
IBM Systems Journal Vol 3 No 2 and 3 1964

4 E R ALEXANDER
A working measured development plan for computer personnel
Proc of the Sixth Annual Computer Personnel Research Conference 1968

Variable topology random access memory organizations

by MARTIN A. FISCHLER and ALLEN REITER

Lockheed Palo Alto Research Laboratory
Palo Alto, California

INTRODUCTION: THE BASIC CONCEPT

One of the most basic of all computer operations is the actual or virtual construction of a data sequence to be used either as the operand for a simple data transfer, or as the argument of a functional transformation. In a significant number of practical situations, the data from which the string is to be constructed are physically scattered prior to the proposed operation or must be scattered after the operation due to physical space limitations or for reasons dictated by the logical structuring requirements of the application.

Powerful software schemes (e.g., the list processing languages) have been developed to deal with the problem of treating scattered data as a contiguous string, but they pay a very heavy price in memory overhead (in some schemes over three-fourths of the available memory is required to handle the addressing mechanisms) and in the processing time required to perform the address arithmetic.

An alternative solution is proposed here which involves the addition of a small **A**ssociative **M**emory (AM) to the addressing machinery of the computer (or peripheral direct access storage device). As will be shown, this hardware modification will permit scattered data to appear contiguous, with only a token overhead cost in memory and processing time.

We note that the data which are to be treated as a contiguous string can frequently be stored as a reasonably small number of physically contiguous strings. Further, the computer operation for moving through a physically contiguous string is the "indexing" operation. That is, a special register adds a fixed constant to the address of the current data element to obtain the address of the succeeding data element. Let us now assume that we wish to treat (individually) physically contiguous strings A_1, A_2, .., A_N as a single string. We can do this by loading a map of the type shown in Figure 1 into an AM which monitors the contents of the effective address register of the computer's addressing mechanism. When an indexing operation results in a match with the search field of the AM, a microprogram interrupt occurs during which the tag field of the corresponding entry in the AM replaces the contents of the effective address register.* The AM is searched in parallel, and essentially no time delay occurs in processing when there is no match.

Since the strings in this example were not specified, they can represent free storage as well as data. Thus, by simply changing the contents of the map loaded into the AM, the topology of the direct access storage device (be it CPU core memory or peripheral storage) can be altered to simplify and speed up the accessing, storage, and processing of virtual or actual data strings.

The VTRAM (**V**ariable **T**opology **R**andom **A**ccess **M**emory) concept presented in this paper will be most effective in those applications in which the number of physical and/or logical** breaks in the contiguity of a data string is small compared to the number of elements in the string, and also does not often exceed the capacity of the supporting AM (**A**ssociate **M**emory). Data structures of this type, where breaks are the exception, occur very commonly.

Logical breaks can be handled by the VTRAM in the same way as physical breaks. However, for some situations we might desire that a logical break be conditional. This can be accomplished by appending an extra control bit to the break addresses stored in the VTRAM that permits address exchange for these

* See Appendix I for further exposition of this concept.

** A logical break in a string is defined here as either an interior entry point in the string, or a "jump" from one interior point in the string to another point in the string.

(ADDRESS OF LAST ELEMENT OF A_1)+1	ADDRESS OF FIRST ELEMENT OF A_2
(ADDRESS OF LAST ELEMENT OF A_2)+1	ADDRESS OF FIRST ELEMENT OF A_3
• • • •	• • • •
(ADDRESS OF LAST ELEMENT OF A_{N-1})+1	ADDRESS OF FIRST ELEMENT OF A_N
SEARCH FIELD OF AM	TAG FIELD OF AM

Figure 1—A storage map, shown without control flags. The order of the entries may be shuffled at will

entries only when the machine condition code is appropriately set.

Implications of a VTRAM for data manipulation

A generalized move operation (CPU to CPU core)

It is frequently necessary or desirable to move a logical data string, unaltered, from one physical location in CPU core to another. Data may be moved from one specific area of core into another prior to bringing in an overlay, or in order to convert a logical data string to a physical string prior to a channel operation. (A single transfer of data from CPU core to a secondary storage medium or output device frequently requires that the data be physically contiguous.) Data may also be moved to reduce the size of the map or the complexity of the address arithmetic necessary to visit the elements of a data string, or simply to reset to zero (or some other fill character) some portion of a data string. While the existence of a VTRAM can eliminate the need for many of these data moves, a significant residue will still be left.

A generalized move can be executed in a VTRAM by simultaneously loading the maps describing both the input and output strings into the AM. The number of bytes to be moved can be specified in the move instruction, or it can be determined by having the AM create an operation interrupt (rather than an address modification) when a match is detected between a specially flagged entry and the current core address. A more detailed treatment of this topic is given in a later section.

Insertions and concatenations from free storage

When storage requirements for a data string are determined dynamically, the area allocated to hold a given string can be exceeded, leading to the necessity of either moving the string to a new (larger) area, or setting up the machinery to handle data chaining. As noted in the introduction, software procedures for data chaining are expensive in storage and processing time overhead. In a VTRAM, free storage is made available to a data string by simply deleting a section of the map of free storage and concatenating it onto the map for the data string after making the initial and final address linkages. It is important to note here that the map representing a data string (in a VTRAM) does not have to be ordered. That is, the ordering of the entries in the map describing the string need have no correspondence with the logical sequencing of characters within the string.

The management of free storage is a recurring problem common to a wide variety of data processing systems. While the VTRAM concept does not significantly alter the nature of this problem, it does permit some simplifications in its resolution. Appendix II discusses this matter in greater detail.

Generalized data paths

It is not unusual to find problems in which the logical connectivity among a set of data elements is more complex than in the simple linear strings discussed previously. Consider the problem of processing a data ring whose elements are physically contiguous. A ring is a string whose last physical element is logically assumed to precede the first; further, the first and last physical data positions have no special logical significance.

In moving through such a structure in a conventional memory, a check must be made after each (address) index operation to see if a jump to the first physical element of the data set is required. In a VTRAM, a single entry in the AM will cause this jump to occur automatically when required.

Perhaps a more important consideration is the case where data stored physically in one configuration must be visited logically along a number of different paths in the course of processing. Without a VTRAM, a rather complex (and time consuming) series of instructions would be required to pick out, in turn, each of the desired paths. With a VTRAM, any given path can be specified by simply loading the map for that path into the AM. Some examples illustrating this discussion are given in the next section.

Channel operations and peripheral storage

The potential application of the VTRAM to the simplification of channel operations is especially significant. The necessity to physically group data (or use some form of channel data chaining, difficult to predict in a dynamic situation) is no longer necessary. The scatter storing of data can similarly be handled in a single operation by the hardware interpretation of the storage map.

To illustrate some of these concepts, let us consider a simplified example.

Assume we have a time-sharing system backed up by a bulk core memory logically partitioned into blocks, each of which is 0.2 the size of CPU core. Thus any load module (represented by a single map) can contain a maximum of five physical substrings. During a swap operation with CPU core, a scatter read (or write) from (or into) the peripheral store can be accomplished by a single channel operation using a five-word AM A later section is concerned with VTRAM applications to peripheral storage.

Patching of "Slow-Write" peripheral storage

One of the potentially useful applications of the VTRAM concept is as an adjunct to optical peripheral mass storage devices. The relatively slow writing times make changing data undesirable, especially for making small corrections or insertions. The logical restructuring of data strings via the VTRAM would permit temporary "fixes." For example, the fix could be stored in a small, fast auxiliary memory until enough alterations have been collected to warrant the generation of a revised "memory plate."

VTRAM data processing

The following terminology will be adopted for the rest of this paper:

- A *string* is a linear sequence of elements, where linearity is a logical concept independent of the physical arrangement of the elements.
- A *block* is a string or substring which is physically contiguous, i.e., the logical ordering of the elements corresponds to their physical placement.
- A *map* is a string of break-transfer addresses which describes the organization of another string; one or more maps can be concurrently loaded in the AM to drive the address-exchanging mechanism of the VTRAM.
- A *file* is a string of maps, and is a unit for loading or storing the contents of the AM.

To present a concrete discussion, let us assume in this section that we are dealing with an IBM System/360 computer equipped with a VTRAM. The VTRAM has a small number of associative memory words (on the order of 10 to 50), each of which has enough bits to represent any valid core address. Corresponding to each word of associative memory, there is a tag memory word which can also hold an address (the "transfer" address) plus several bits of control information. One such bit is the "end of string" bit which signals to the VTRAM that this break address constitutes the end of a data string. A second is the "end of file" bit which denotes the end of a string of maps. (The VTRAM may contain several such files at one time, each file typically containing one or more maps, each map describing a string of data.) Finally, two or three bits can be used as a map key.

The VTRAM, of course, has the property that it is searched for a match in parallel, and essentially no overhead is incurred when no match is found. The overhead in replacing the content of the register being monitored (the "effective address register") by the content of the tag field of the VTRAM is also negligible. The associative memory will only have to respond to an "equal" condition, thus reducing the cost of the unit. In addition, the associative memory will have the property that on being loaded it will use the first available nonzero registers; a property usually obtainable at no extra cost and which is very useful, as we shall observe.

Some additional IBM 360 instructions

The following commands have to be added to the CPU repertoire to manipulate the VTRAM:

1. *ON/OFF Associative Memory*
2. *Clear Associative Memory* (*CAM*). This will replace the content of the associative memory registers by zeros. A selective clear based on the map key can also be requested.
3. *Load Associative Memory* (*LAM*), from core location T. T is the starting location of a map. Note that this map automatically orders the stroage and does not itself have to be ordered; the order of occurrence of the break-transfer addresses in the map is of no consequence. This is a very important property of the VTRAM, as it greatly facilitates additions and deletions to the map.

   The loading of the associative memory continues until an end-of-file bit is encountered in one of the entries. If the available associative memory is exhausted before the end-of-file indication is reached, an interrupt occurs (or a condition code set) and the address of the last entry loaded is posted. The possible remedies to this overflow condition will be discussed. Note that the VTRAM can be loaded with several maps corresponding to several logical data strings, at the same time. This facilitates storage-to-storage operations.

A highly useful variant on the LAM instruction is a BLAM (**B**ackward **L**oad **A**ssociative **M**emory), in which the break-transfer address pairs, as they appear in the table, are interchanged when loaded into the associative memory. This facilitates the backward scanning of a logically contiguous block or string of data.

4. *Store Associative Memory* (*SAM*), used by the supervisor during interrupts.

The augmented MVC command

To make use of all of this machinery, the storage-to-storage instructions (SS) of the 360 will be assumed to operate using the VTRAM. This means that the length specification will become unnecessary, since a string-end condition is now explicitly represented by break address entry with a flag.

A typical sequence of commands for, say, moving data from string A to string B (the strings possibly consisting of several blocks each) would be

CAM	Clear the associative memory (optional)
LAM TA	Load the map which describes string A
LAM TB	Load the map of string B (note that this is in addition to TA)
MVC B, A	Start move operation, with the initial addresses being A and B. (Note that the VTRAM contains only the break addresses, and not the initial addresses of the strings.)

The MVC instruction will terminate as soon as an end-of-string indicator is reached for either string A or string B, returning a condition code corresponding to each applicable case. An alternate possibility, in case the end-of-string for A is encountered first, is to fill the rest of B with, say, zeros.

Matrix multiplication

As an example of generalized data paths, we will look at a procedure for forming a product of two matrices, C = A*B. The three matrices are not assumed to be located in contiguous storage, but may rather be scattered all over core. We will however, assume that each *row* of the A and B matrices is in one block. (The matrices are n by n, with each element being a half-word integer, so that each row is 2n-bytes long.)

```
              CAM                First clear the associative memory
              LAM   TA           Load the maps for the matrices A, B, C
              LAM   TB
              LAM   TC
              LA    RA, A        Register RA points to A(1, 1)
              LA    RC, C        Register RC points to C(1, 1)
              LH    R1, n        R1 is the counter for the I loop
LOOPI         LH    R2, n        R2 is the counter for the J loop
              SR    R6, R6       R6 has displacement from start of row for B(K, J) for J = 1
LOOPJ         SR    R4, R4       R4 has the partial sum for C(I, J) = Σ_k A(I, K)*B(K, J)
              LA    RB, B        Register RB points to B(1, 1)
              LR    R7, RA       R7 is the pointer to A(I, K) for K = 1
              LH    R3, n        R3 is the counter for the K loop
LOOPK         LA    R5, 0(RB, R6)  Pointer to B(K, J)
              LH    R8, 0(R7)    A(I, K)
              MH    R8, 0(R5)    B(K, J)
              AR    R4, R8       Partial sum
ENDLOOPK    * LA    RB, 2n(RB)   Step row address. Note that if a break address is reached, the new
                                 transfer address gets inserted into RB from the effective address
                                 register. Thus, the next time through the loop, when the LA R5,
                                 0(RB, R6) instruction is executed, the correct address will appear
                                 in R5
              LA    R7, 2(R7)    Move pointer to A(I, K) to next element in row
              BCT   R3, LOOPK    Go through the k-loop n times
```

```
ENDLOOPJ  STH    R4, 0(RC)      Store C(I, J)
        * LA     RC, 2(RC)      Step pointer to C(I, J). Again note that if a break address is en-
                                countered, the transfer address replaces RC. (This works even if
                                breaks occur within a row in the C matrix)
          LA     R6, 2(R6)      Step element displacement from start of row for B(K, J)
          BCT    R2, LOOPJ      Go through the J-loop n times
ENDLOOPI * LA    RA, 2n(RA)     Step row address for A. Again note the automatic substitution of the
                                transfer address
          BCT    R1, LOOPI      Go through the I-loop n times
```

Because of the VTRAM device, this program did not differ substantially from any matrix multiplication program for contiguously stored matrices. Of course, similar procedures are used on computers not equipped with a VTRAM. These, however, require that the starting row address for *each* row of A, B, and C be stored in a table; with the VTRAM, only the actual break addresses need to be stored. An additional advantage obtained by the VTRAM is that the C matrix can be scatter-loaded randomly, without the restriction that no row can be broken.

Note that in the example it was necessary to return the break address modifications to one of the index registers used to control the addressing sequence because the data path was traversed in a sequence of instructions under program control, rather than as a single instruction triggering a micro instruction sequence. This was accomplished by means of the LOAD ADDRESS instruction which sums a fixed displacement and up to two index register values in the effective address register and then returns this sum (in our case, to one of the participating registers). When the effective address computed in this process matches a VTRAM break address, the effective address is replaced with the corresponding tag address, and this address, in turn, is transmitted back to the index register specified as the receiver for the LA instructions.

Another example with logical data path

As a second example involving generalized data paths, consider the problem of analyzing the contour of a two-dimensional graphical object, which is represented in storage by a series of fixed-length records, each giving the (x, y) coordinates of a point on the contour together with some additional information such as the identity of the line segment to which the point belongs. (The contour is partitioned into a series of line segments which are physically contiguous substrings.) The order in which the records are stored corresponds to the sequence in which the points occur on the contour of the object, with some arbitrary break point to permit storing this file structure, which is logically a ring, as a (physical) linear string. We assume here that there are no physical breaks other than the one which closes the ring.

In one operation we might prefer to start with the point having the greatest coordinate value, and count the number of points for which the coordinate monotonically decreases. Without a VTRAM, after each point is examined, we must check for the ring-closing physical break address in the table before indexing to the next point. With a VTRAM, storage of the break address would cause an automatic return from the end of the string back to the beginning if an attempt was made to index past this last entry in the file.

In a second operation, we might prefer to find the center of gravity of some selected subset of line segments. Without the VTRAM, we would be required to check the segment ID of each point against the list of desired segment numbers before including the point in the computation; or, after each index operation, a check would have to be made against a list giving the starting and terminating address for each of the desired line segments; or some special coding would have to be written to selectively find the desired points (based on address) without search. Regardless of which of these methods was chosen, a certain amount of data-dependent code must be written and stored, and the execution of this code must pay a time penalty to continually search for the logical break addresses. With a VTRAM, once the logical break addresses for the selceted line segments has been loaded in the AM, the operation can proceed as though only the selected line segments were present. No data-dependent code is required and no time penalty is accessed for searching for the required points.

The overload problem

When a map for a given string gets larger than the available number of associative memory registers, it has to be partitioned so that the consecutive partitions of the map (submaps) now correspond to the logical ordering within the string. Given such an organization of the map, only a part of it need be loaded into the VTRAM at one time; if a given operation is not completed before the last break address corresponding to the end of a submap is reached, the next submap replaces the old one, and the operation is repeated. This process continues until the end-of-map indicator is encountered.

When a large map is thus partitioned into submaps, each of which is small enough to fit into the VTRAM, it should be noted that there is no need for the submaps to be ordered internally. This fact facilitates the sorting process, and also allows subsequent insertions of new break-transfer entries at any available position within the appropriate submap.

Note, however, that even the partial ordering of a map entails what is usually a very large overhead, and the VTRAM should contain enough registers to obviate the necessity for this ordering in the great majority of cases. If it were not for this fact, we could always assume that the maps are ordered, and would be able to operate using an associative memory with only one register; i.e., any ordinary register which can monitor the effective address register would suit our purpose.*

The reordering operation may itself make use of the available associative memory. The use of associative memories to facilitate sort operations has been studied in the literature.[1]

Some limitations

While the VTRAM mechanisms presented here are well suited to the separate handling of logical or physical breaks, the combination of such breaks in the same string will cause special problems. To specify a direct jump from the i^{th} element of a string to the j^{th} element across a physical break, some address arithmetic using the string storage map must be performed. The arithmetic is simple and needs to be performed only once. Nevertheless, the computation is time consuming and sometimes cannot be done at the time the string is stored, but must be made when the jump is required. For these reasons, an executable instruction string which contains many dynamically determined branches is not a desirable candidate for VTRAM-controlled scatter loading.

Using a VTRAM for peripheral storage management

A key problem which arises in many different contexts in dealing with random access storage devices is the fact that there is really no such thing as a truly random access device; all existing devices are actually organized into groups of records, each one of which is bounded on either side by other records. This causes problems when the information contained in a record changes dynamically in size; for example, it may no longer fit into its former place. One is now typically faced with a decision of whether to invest the time in trying to make it fit into one place (either by pushing aside its neighbors or by finding a different place for it which is large enough), or to string it out in several records by chaining the various pieces, thereby sacrificing time during retrieval. A VTRAM can be used here to great advantage to create logical linkages between physical records which are not physically contiguous, thus allowing them to be written or read using but one CPU channel command; i.e., to make them look like a single physical record to the computer.

We will discuss two classes of random-access storage devices; a bulk core memory, and a fixed-head rotating memory. In both cases we shall assume that the CPU communicates with a controller unit for the memory in question, and that this controller is in reality itself a small stored-program computer; it is this small computer that we wish to equip with a VTRAM in order to facilitate the restructuring of records without actually moving data around. It should be pointed out that the controller will be able to perform a certain amount of housekeeping operations "for free" if it can do them subsequent to the completion of a given I/O operation, since there will be periods when the CPU is not using this class of devices. If the controller can schedule housekeeping operations, for example by waiting a certain time interval to give the CPU a chance to start a second operation which might have been present in a queue, more efficiency can be achieved. Thus, we can in general tolerate more overhead for housekeeping in the controller than in the CPU. On the other hand, for a rotating device, the mechanical aspects of the memory sometimes present critical timing problems for the controller.

Drum memory

To lend concreteness to our discussion, we shall describe a specific rotating memory. This is a drum, with 800 tracks, each of which is somewhat over 4096 bytes long (the exact length will depend on the particular memory organization that we choose). The drum is rotating along the tracks, so that a given track can be read during a single revolution. Only one head can be reading or writing at any given time, but heads can be switched at any time at electronic speeds; hence,

* It might be interesting to use this approach as a first approximation to a VTRAM. Note that this monitoring function of the effective address register (by switches which are manually set) is already present in the IBM 360. The hardware modification required to make this register addressable from within a program or a microprogram would seem to be fairly minor.

during a single rotation data can be picked up from several tracks, but necessarily from different positions along the tracks.

We shall assume that the drum is equipped with a timing track, whose current content together with the current track indicator are being read into a register which can be monitored by a VTRAM. (This register will serve the same function as the "effective address register" already discussed.)

Along each track, memory will be divided into sectors. A sector is a quantum of storage, say 64 bytes long; thus our drum has 64 sectors per track. Since, by definition, a sector is the smallest addressable unit of storage the timing track need only carry information as to the current (or next) sector.

Becuase of the possible time lost in head switching, there may actually have to be a physical separation (amounting perhaps to several byte positions) between the end of a sector and the start of the next one.

The first part of each record consists of its storage map. (See Figure 2.) The record is addressed (by the computer) by giving the address of the sector which contains the map. The map is read into the VTRAM in the controller; it then controls the mechanics for reading the rest of the record by causing head switching to take place. If the map terminates in the middle of a sector, the data portion of the record starts immediately. If only retrieval is desired, the data transmission to the computer need not commence until after the "end-of-map" indicator has been read into the controller. For a write operation, the computer will have to supply not only the address of the start of the record, but the map as well. The onus of storage management on the drum, and the decisions as to where to put the various pieces of a record, must be left up to the central processor. The controller is capable of performing this function equally well, but the efficiency of use of a rotating memory depends critically upon the scheduling of accesses to the memory.[2] To do intelligent scheduling, the controller would have to have access to a much larger body of dynamically changing information than is normally feasible (except on computers like the CDC 6600, where the controller is itself a full-fledged CPU with full access to all of core).

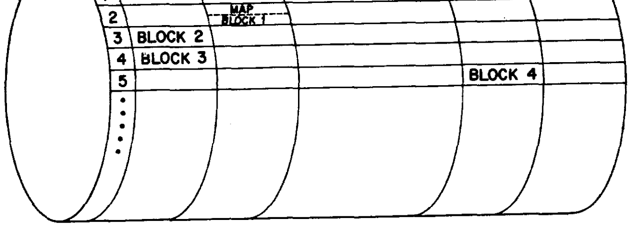


Figure 2—The physical organization of a record in storage. The address of the record shown, from the point of view of the central processor, is Track 1 Sector 2, which is actually the address of the map

Let us look at the case of a record which is being rewritten, but which requires more space than it formerly occupied. Instead of releasing the record's current storage to the free storage pool and then allocating a contiguous block of storage for the entire record, as might be done on devices without a VTRAM, the drum storage allocator of the CPU would be asked to allocate additional storage from a longitudinal position which begins immediately following the termination of the last sector in the current record. (Various alternate strategies are possible if such storage is not currently available.) That is, a piece is simply added on at the end of the existing record, thus making a longer record. Note that this piece can, in our case, come from a selection of any one of 800 different sectors.

The allocation procedure has the very important property that the new storage block is always added on at the logical end of the current storage. What this means is that our maps are always ordered—successive entries in the map represent successive blocks of the record. Thus, there is no need to use the entire break-address portion of the map to monitor the timing track. Instead, the controller can have a pushdown stack, in which only the top element of the stack is used to monitor the effective address register. When a break address is encountered, the next element (break-transfer address pair) is popped up and becomes the new top of stack.

A basic underlying assumption in this discussion is that the entire record is rewritten during an output operation. This distinguishes the use of the VTRAM device in the present case from its use for core-to-core operations as discussed in a previous section, where a logical unit of data could be restructured dynamically in many ways other than addending at the end.

To recapitualte our proposed method for handling drum storage: each record carries along (at its beginning) a map of where the record is located; this map is loaded into the VTRAM device and is used for automatic head switching as required. There is essentially no overhead lost in retrieval time in this scatter storage of a record compared to having the entire record stored on a single track. There is, however, some overhead paid in storage. For our drum, each map entry would consist of three 8-bit bytes: one to monitor the timing track (for 64 possible different sectors) and two required for the proper head selection plus some miscellaneous bits. In the worst possible case, where a head change is required for each sector, the storage overhead is somewhat under five percent.

Bulk core memory as peripheral storage

This type of storage device is characterized by the fact that there is no latency time due to rotational delay to worry about, as was the case for the drum. Thus, as far as the computer is concerned, all storage locations are equally accessible, and it does not matter where various pieces of data are stored. As in the case of the drum, however, we will assume that the computer deals with storage in records, and that a single record is to be fetched or written with a single channel command. It is, of course, possible to assign much more complex structure to the data, and have this structure reflected in the nature of the computer-controller communications. As this would unnecessarily complicate the ensuing discussion without contributing anything, we will assume the simplest possible structure in the bulk core.

Using a VTRAM enables us to have a record consist of many noncontiguous blocks. For bulk core, all free storage management can be left up to the controller. We propose a similar record format as for the drum, where the computer addresses a "header" which contains a map of the record; the computer, however, need never see the map. The free storage handling strategies discussed in Appendix II for CPU core are equally valid for the bulk core as well. The significant differences for bulk core are (1) more time can be invested in periodic recondense operations since they can be scheduled "off-line" by the controller, and (2) the VTRAM implementation need not actually involve any associative memory, but only a pushdown stack.

When the CPU initiates a bulk core write operation, the free-storage-map (held permanently in a pushdown stack) is used by the VTRAM to direct the store operation. The top of the free-storage-map, which corresponds to the storage blocks needed to contain the transmitted record, now becomes the map for this record.

In the case of the bulk core, the most convenient location for the map describing a record is immediately following the record. However, since the map itself is a record which may have to be scatter loaded, the initial segment of any such map must contain the break addresses needed to retrieve the map. (This initial segment terminates with a special flag bit.) Thus, after a write operation has been completed, the controller adds to the top of the map for the stored record a prefix (possibly null) of additional break addresses and this augmented map is stored (under control of the VTRAM and the map prefix) immediately following the record it describes. Finally, the address of the first entry of the prefix is returned to the CPU as the address to be used in reading the record. This address is available for returning to the CPU immediately at the conclusion of the write operation, even though the contents of this location will not be determined until after the map itself has been written.

For a read operation, the CPU-supplied address is used to obtain the map prefix which then directs the loading of the map itself into a VTRAM pushdown stack. Now the map directs the requested read operation.

Comparison with other address-mapping schemes

Associative registers and other special-purpose memory addressing hardware are currently employed in a number of computing systems (e.g., IBM 360/67, GE 645, B 8500)[3,4,5] to implement address mapping for paging and segmentation schemes. These concepts and their associated hardware organizations are significantly different from the VTRAM concept presented here. Paging is a scheme for making the fast core appear larger than it actually is (virtual memory), and is accomplished by defining a mapping function from a large virtual space into the small physical space. The paging hardware is employed at every memory reference and at least one conversion per reference is required to translate the symbolic address contained in the instruction into a physical address. Pages of core are usually of a fixed size, although they may come in several sizes. This organization allows jumps into the middle of a page to be handled easily. Maps (segment and page tables) pose a special handling problem and entail storage overheads.

The VTRAM concept is concerned with dynamically restructuring the logical contiguity relations between data within the same real space without having to move the data. A second application of the VTRAM concept is to reduce the programming and timing requirements for moving physically scattered data between a peripheral device and core, or from one set of core locations to another. The VTRAM hardware is used to exchange one physical address for another only at the break points specified by the active storage maps; otherwise it does not intervene in the addressing process. The length of each block of stroage is of no consequence. The VTRAM is intended to support operations which index through the data, and unexpected logical breaks can be handled only with difficulty. The storage maps used by the VTRAM can themselves be scatter loaded just like any other data string; this permits a saving in space, for even though the maps may dynamically change in size, no contiguous blocks of storage need be reserved for them.

CONCLUSIONS

The VTRAM concept, as developed in this paper, is a hardware organization for achieving, on almost any digital data processor, a string processing capability, extending to channel and peripheral operations, at very low cost. The central idea is that of "address exchange" when a critical boundary is crossed during an indexing operation. Implementation is achieved by storing these boundaries in a small associative memory so that many of these boundaries can be searched for in parallel, thus avoiding any significant processing time overhead. Given the presence of the AM, it is also available for use in a more conventional manner.

The major advantage gained via the VTRAM is the ability to do "string processing by exception" for strings which contain relatively few breaks.

Such strings (the authors feel) are very common, and come about as a result of insertion, deletion, or rearrangement operations on formerly contiguous data. They also arise when contiguous data, stored physically in one configuration, must be visited logically along one or more different paths in the course of processing. There are types of strings, however, such as those consisting of executable code, for which the VTRAM mechanization as described here is not very useful.

One area where the authors feel the VTRAM concept has much potential value is in augmenting data transfer machinery.

It is impossible to give a simple answer to the question of how much core is required to adequately support a given CPU, and similarly the question of how much AM is required to optimally implement the VTRAM concept with a given amount of core cannot be answered except in very specific situations. The authors feel that there are many significnat applications where a small fixed AM, say 10 to 50 words, would be very valuable regardless of the amount of core.

For the VTRAM concept to be useful, the AM size must be large enough to contain the maps generated by the various applications. This will be ensured for most cases by free storage management procedures and by programming limitations dictated by the actual size of the AM. In the unusual case of severe fragmentation of a given storage area, causing the AM size to be exceeded, several procedures utilizing the VTRAM are suggested.

An especially important advantage of the VTRAM concept is its applicability to almost any reasonably sized computing system with very little hardware modification.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Professor Harold Stone of Stanford University, and to others who read the preliminary draft, for some very cogent comments and ideas which contributed to this final paper. Thanks are also due to Margarett N. Collins for her able editing of the manuscript.

REFERENCES

1 R R SEEBER A B LINDQUIST
Associative memory with ordered retrieval
IBM J Res Dev Vol 6 January 1962 126–132

2 P DENNING
The effects of scheduling on file memory operations
Proc S J C C 1967

3 E L GLAZER J F COULEUR G A OLIVER
System design of a computer for time-sharing applications
Proc F J C C Part I Vol 30 1965 197-202

4 J D MCCULLOUGH H K SPEIERMAN F W ZURCHER
A design for a multiple user multiprocessing system
Proc F J C C Part I Vol 30 1965 611-617

5 W T COMFORT
A computing system design for user service
Proc F J C C Part I Vol 30 1965 619-626

6 D E KNUTH
The art of computer programming, vol 1: Fundamental Algorithms
Addison-Wesley 1968

APPENDIX I: VTRAM IMPLEMENTATION

Figure 3 shows a typical computer data flow organization augmented by an AM. The main point to note here is that implementation of the VTRAM concept can be accomplished with minimal disturbance to the conventional data flow paths. The essential requirements are a connection to the main memory output bus to permit the loading of the AM, and a connection to the Address Adder (or "effective address register") so that monitoring and address modification can be accomplished.

To more fully utilize the capabilities of the AM, it may be desirable to introduce additional direct paths to one or more of the index registers. However, all of the operations discussed in this paper can be carried out without these additional connections.

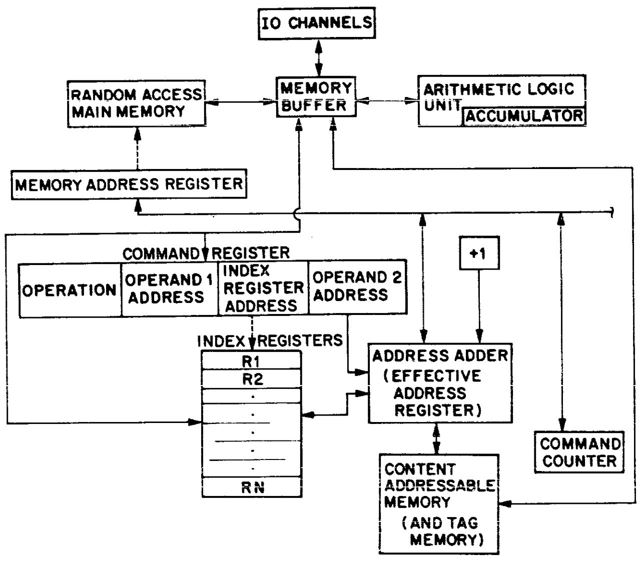


Figure 3—Typical computer data flow organization augmented by an associative memory to realize a VTRAM capability

APPENDIX II: STORAGE MANAGEMENT STRATEGIES USING A VTRAM

A major function of the VTRAM device is to facilitate the dynamic restructuring of data, without actually having to move the data about from place to place. A very important application of this concept is in storage management for complex dynamic systems, such as a time-shared executive system. A rough description of the environment might be as follows. (This is a cross section at time t_0.)

- There is a collection of storage called "free storage" which is noncontiguous.
- There are n jobs, each of which occupies some noncontinguous region of storage. These regions are mutually disjoint, and do not intersect free storage.
- Each job may request additional storage from free storage. The request will be for a certain amount of storage, and may be allocated (by the executive) at its discretion from anywhere in free storage. When such an allocation is made, the storage area is deleted from free storage and becomes part of the job's storage region.
- Each job may release any or all of its storage to free storage.

It should be pointed ouf again that a VTRAM is not particularly useful tor executing scatter-loaded code, as program strings tend to have too many logical breaks (interior entry points). The reader should assume for this discussion that the jobs are requesting core for data manipulation. For example, a job may wish to read in a record from a peripheral storage device, insert a new field, and write the record back into the storage device. Note that this operation is precisely of the form discussed above: only one or two breaks are introduced into a (formerly) contiguous data string.

Let us discuss the free storage management function. This management must perform two distinct functions: that of allocation and that of releasing storage back to free storage.

The management of free storage is concerned with creating a balance between the tendency for blocks of storage to become progressively smaller (through the randomizing action of the allocation-release process) and the overhead involved in rebuilding larger blocks from the available fragments.

The overhead for condensing free storage can be paid in a number of different ways, involving such considerations as the time required to allocate or release storage, the frequency with which the condensation must be repeated to keep the average block size above some minimal value, and the complexity of the hardware and algorithmic procedures needed to perform the condensing operation.

Assuming, as we are doing, that each job will manipulate its own data structures using the VTRAM, which has a limited storage capacity, it is incumbent upon the executive to minimize the storage fragmentation during the allocation function; otherwise, each job will have its storage broken into so many noncontiguous pieces that the storage capacity of the VTRAM will be exceeded very often, with a resulting high overhead, thus negating the benefits from having this device available. On the other hand, both the allocation and the releasing function are performed so frequently that if the system is to function efficiently, these functions must not take too much time.

The strategy discussed below is derived by adding a VTRAM to the commonly used "first fit" strategy; e.g., see KNUTH, Section 2.5.[6]

Free storage is represented as one string (noncontiguous), with a map of break-transfer addresses. When a request for n storage units comes in, storage is "peeled off" from the top by assigning an many blocks as are required to satisfy the request. Since the end of the request will typically fall in the middle of a block, a new break-address will terminate the block given to the job, and a new start address will be assigned for free storage. This allocation operation is extremely fast.

The storage release operation itself is equally simple. The job denotes the area it wished released by supplying a map; this map is addended to the free storage map and the operation is finished. (The reader may note that this is conceptually identical to a "threaded list" organization of free storage.) Since the area required for the free storage map itself necessarily has to be finite, the VTRAM can be used to handle this area in a cyclical fashion (with a fixed maximum size for the number of entries). Thus, as entries are taken off the top and added at the bottom, the two operations are eventually performed at the same rate, and a steady-state cyclical storage area suffices.

The disadvantage of this strategy is that storage will become increasingly more fragmented, since no attempt is made to find the best blocks for this particular request. If this approach is to be made workable, a "garbage collection" operation must be performed periodically after storage is released.

To facilitate the garbage collection, it is useful to represent a storage map as an (unordered) collection of ordered triplets (B_i,T_i,AL_i), where B_i and T_i are the break-transfer addresses as before, and AL_i is the address of the last element of the block started by T_i. That is, every AL_i is equal to some B_j in a one-to-one fashion. In addition, the start of a block is represented by a triplet (0,T_0,AL_0), where the break address is empty. The reason for this redundancy is that the break-transfer addresses do not lend themselves to a convenient identification between a T_i which starts a given piece and a B_j which terminates it. Of course, during the use of the VTRAM, only the B_i's are loaded into the associative memory, and only the T_i's need appear in the tag part.

To return to the garbage collection algorithm: The operation of combining contiguous blocks consists simply of looking for matches between the AL_i's and the T_j's. Anyt'me that a match $T_j = AL_i$ occurs, we simply replace AL_i by AL_j and delete the triplet (B_j, T_j,AL_j) from the map. Most of the overhead in this operation is in the search operation; a function greatly facilitated by the presence of the associative memory.

After the garbage collection, free storage has been completely condensed and consists of a number of noncontiguous areas, each of which is represented by an entry in the map. Note, however, that the ordering of the blocks is totally random, since there is no advantage to be gained in sorting these blocks on their respective core addresses; nor is there anything to be gained in sorting them by size, since the allocation strategy calls for assigning storage from the top of the free storage list.

If the associative memory is large enough to accommodate the entire T_i vector, the loading operation during the search for matches needs to done only once, thus speeding up the condensing process considerably. It might therefore pay to do the condensing quite frequently, in order to keep down the size of the T_i vector (as well as to cut down on the fragmentation of the allocated storage). The exact tradeoffs involved (i.e., what is a reasonable size for the associative memory, and how frequently should one recondense, given certain assumptions about the job mix and probability distributions for storage request-release operations) will be the subject of a simulation study conducted by the authors, with the results described in a forthcoming paper.

In summary: this strategy for storage allocation and the corresponding representation of storage blocks is characterized by extremely rapid handling of requests and releases. The storage representation is in a form directly utilizable by the jobs because of the presence of the VTRAM device. Some overhead is incurred for the periodic recondense operation associated with storage release, but this overhead is considerably reduced by the presence of the associative memory. The overall efficiency of this method will depend upon the size of the associative memory; the exact relationship is unknown pending the outcome of a simulation study.

Fault location in memory systems by program*

by C. V. RAVI

University of California
Berkeley, California

INTRODUCTION

The subject of automatic fault location in memory systems by program so far has been neglected in computer literature. A program (AMNESA) has been written at Honeywell that not only detects failures in memory, but also diagnoses the cause of the failure. This paper describes the approach used in the writing of AMNESA. It is also shown that this approach can be used for different memory organizations.

In the past, memory diagnosis has depended considerably on the intuition and experience of the Field Engineer. Standard test routines consist in moving information into and out of memory, comparing and printing out the location of the error and the faulty information. After this the Field Engineer was left with the problem of finding the cause of the errors. Thus, emphasis has been on the detection of errors. AMNESA goes further to isolate the cause of the errors.

Objectives

The diagnostic program was written with the following objectives in mind.

1. Special hardware which may be necessary in order to run a diagnostic should be kept to a minimum.
2. The diagnostic should isolate the cause of failure to one or two circuit packages.
3. Multiple failures should not confuse the diagnostic.
4. Very little should be left to the Field Engineer — to minimize downtime.
5. Failures in the parity plane circuits should be diagnosed.
6. Failures in the Memory Address Register should be diagnosed.

By and large, AMNESA has achieved all these objectives.

Necessary hardware features

Something can now be said about hardware, in general, necessary for the running of a memory diagnostic on a computer with a malfunctioning memory.

1. The diagnostic program has to be loaded into an area of memory which is working correctly in order to diagnose a malfunctioning area.

One can sub-divide the memory in most machines into modules between which there is very little or no common circuitry. Assuming that all modules do not fail simultaneously, and that the module is large enough to contain the diagnostic, a diagnostic can be loaded and run. Needless to say, the program should be relocatable from module to module. On many of the older machines, index registers were contained in main memory. In such cases, the index registers themselves must be relocatable or the program should be written without using index registers. In the H-1200 and H-2200 computers, the module size is 32K characters and AMNESA is considerably smaller.

In computers that have interleaved memories, the interleaving is usually between modules. The simplest way to guarantee enough room for the diagnostic is to have a hardware feature that allows the addressing to be sequential from module to module. Such a feature is, of course, very easy to implement.

If it is desired, however, that as little of memory as possible should be destroyed by the running of the

* The work described in this article was done while the author was with Honeywell E.D.P., Waltham, Massachusetts. Present address: Institute of Library Research, Univerisity of California, Berkelery, California 94720.

diagnostic, other techniques can be employed. First consider fixed word-size computers with n-wayed interleaving between (word-oriented) modules. In order to run the diagnostic in any module there are two requirements.

a. Force the assembler assembling the diagnostic to assemble instructions and data at word addresses that are multiples of n. This is quite easy as it just entails stepping the location counter in the assembler by n rather than by 1 as usually done.
b. One also requires a feature by which the program instruction counter (in the hardware) is stepped by n rather than by 1.

In the case of variable instruction size and variable data size machines, one would have to restrict the program to be written in instructions that are smaller than or equal to the word size, of each module. In exceptional cases, this may not be possible. Memory restructuring would then be necessary. Similar restrictions would apply to the kinds of data such a program could handle. During execution, however, there is an advantage in such machines. The Program Status Word (or equivalent) in such machines has an Instruction Length Code field (ILC) which is added to the program counter to reference the next sequential instruction. This ILC field could then be forced to n and the execution of the program could be effected.

The above is to show that the problem of interleaving is not insurmountable and in any specific case probably quite easy to solve—both technically as well as economically.

If a very short memory diagnostic program can be written, it may be worthwhile to have a Read-only memory (that can be used in a special mode of operation) that contains the diagnostic. This may turn out to be quite reasonable in a microprogrammed machine.

2. A parity error in the area of memory being tested by the diagnostic should be signalled to the program, while a parity error in the area into which the diagnostic is loaded should cause the computer to halt.

The signalling can be done by an internal interrupt or fault which indicates a parity error has occurred. Since initially it is not known where the fault is, the diagnostic has to be loaded into different modules until a particular module that is functioning normally is found. In the H-1200 and H-2200 computers, if a parity error occurs in a protected area of memory, an internal interrupt is generated and the program does not halt. If the error occurs in an unprotected area, the program halts and the parity error indicator lights up on the console. AMNESA itself takes care of protecting the module it is testing. Of course, the module into which AMNESA is loaded is unprotected.

The superficial solution of disabling the parity generation and check circuitry neither allows the operator to determine if the program is running normally nor does it allow the diagnosis of the parity plane.

3. Although not absolutely necessary, it may be advisable to have a diagnostic mode—in addition to the supervisor and user modes—so that special conditions can be treated.

Fortunately no hardware changes were required on the H-1200 or H-2200.

Description of the memory structure in the H-1200, H-2200

The basic unit of information is a character. The H-1200 and H-2200 are variable word-length computers. Each character in memory consists of 8 bits plus a parity bit. Both machines have similarly organized memories.

The memory is a fairly typical coincident current memory. Each plane consists of 16K (1K=1024) bits. There are 9 planes in a stack. Two stacks constitute a drawer of 32K 9 bit characters. The memory is modular with respect to drawers of 32K, i.e., there is very little common circuitry between drawers. Even within a drawer, some circuits are common to two stacks while others are not. A block diagram of the memory is shown in Figure 1.

Circuitry associated with the memory can be classified into three types;

a. Memory Address Register (MAR) Circuitry
b. Character Circuitry
c. Bit Circuitry

The memory address register contains the address of the location to be accessed. The register is 18 bits long allowing a maximum memory capacity of 262K characters.

Circuits that are concerned with the selection of a certain location (character) are classified as character circuitry. Failure of these circuits will result in errors in more than one bit of a character (theoretically all bits of a character).

By bit circuitry is meant circuits that are predominantly concerned with the bits within a character. The failure of any of these circuits will affect a particular bit within a character.

In addition to the circuitry associated with the memory, there are windings in the stack that could open up. Examples of these are X-lines, Y-lines, inhibit lines, and sense lines. There are 8 inhibit lines and 4 sense lines per plane.

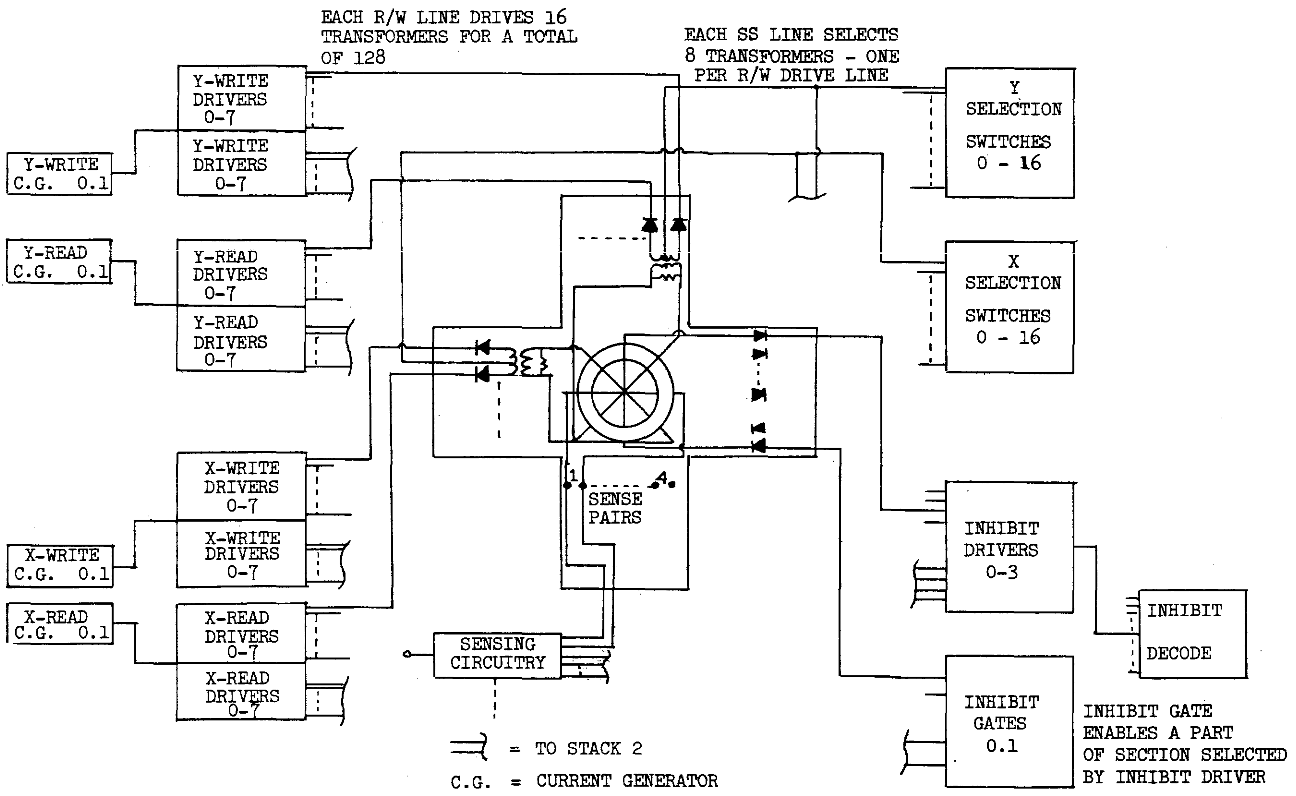


Figure 1—Block diagram of H-1200, H-2200 memory

Types of failures

The types of failures that are diagnosed by AMNESA are the following:

1. Circuit Failures
   a. Output always high
   b. Output always low
   c. Output inconsistent (e.g., faulty sense amplifiers)
   d. Input gating diode or diodes open

   Shorted diodes are not considered when the resulting errors are unpredictable. Fortunately, diodes usually open and cases of shorted diodes are extremely infrequent.
2. Open (cut) windings
3. Core Failures
4. Memory address register failures consisting of bits stuck to "0" or "1."

Testing procedures

Before one can diagnose failures, one has to be able to detect them. In other words, information has to be read into memory and then read out. Much work has been done into the nature of the words that should be utilized. There exist patterns for generating worst-case disturb voltages on sense windings, patterns to heat up the cores and so on. In our case it is usually enough to write and read three different words. The first is a word consisting of all binary 1's and the second its complement (all 0's). Unfortunately, on any memory using odd parity on an even number of bits (as is normally done), the parity bit is the same; i.e., 1 for the first two words. Therefore, for the diagnosis of the parity plane another word which forces the parity bit to 0 should also be used. AMNESA tests the locations using three different words as described above.

Diagnosis

Temporarily removing the restriction to memories, it is obvious that, for the testing and diagnosis of a system, certain test points have to be made available to the program. The term test point is used in a very general sense; i.e., any internal part of the system that can be monitored. These test points generally consist of operands, indicators, and status information. For memories, the only test points required are those which are anyway available—the bits in the memory.

Define any location in the memory as bad if the information read from that location does not match the data written in. Any fault in the memory system (with one exception) can be visualized as partitioning the memory into two classes consisting of the addresses of the locations that are bad, and the other corresponding to the good locations. Let us call such a partitioning a pattern. Naturally, the pattern can be identified uniquely by considering either class. Since every fault creates a pattern, our problem of diagnosis reduces to observing the pattern and identifying the fault. If all faults created unique patterns, the diagnostic would achieve Objective 2 perfectly.

If several faults cause identical patterns, the resolution of the diagnostic (the number of p.c. cards that have to be tested by other means in order to find the faulty card) suffers.

The exception referred to above occurs when a fault occurs in the memory address register (MAR). Normally there is a one-to-one correspondence between the n-tuples (addresses) in the MAR and the 2^n locations in memory. When a bit in the MAR fails, this correspondence does not hold any longer and the true correspondence holds only for 2^{n-1} addresses. Addresses loaded into the MAR are in some cases translated into other addresses. No parity errors will occur and a special algorithm is required.

1. *MAR diagnosis.* When a system has several modules, there is usually a local MAR for each one. If any local MAR fails, the diagnostic can be loaded into another module. If the main MAR fails, this part of the diagnostic should be in a separate memory (such as a read-only memory) or should be executed manually. One method of MAR diagnosis is to write the address of each location into the location and then read sequentially from the bottom of memory. This algorithm only indirectly aims at diagnosis. The following algorithm is considerably better.

Let the MAR consist of n bits numbered 0,1 . . . n-1 from the least significant side.

(i) Write into locations $[(2^n\text{-}1)\text{-}2^i]$ where $i = 0,1, \ldots n\text{-}1$, some word.
(ii) Write into location $(2^n\text{-}1)$ some word X.
(iii) Read from the locations in step (i) using the algorithm that if location $[(2^n\text{-}1)\text{-}2^i]$ contains X, bit i is bad where $i = 0,1 \ldots n\text{-}1$.

The operation of this algorithm is not as straightforward as it looks. An example will clarify the algorithm (see Figure 2).

Let the MAR consist of 4 bits. Assume there are two errors—bit 0 is stuck to "0" and bit 3 is stuck to a "1."

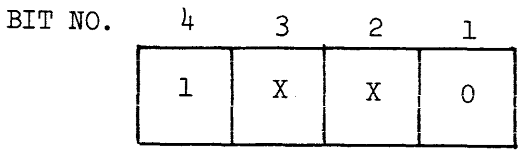


Figure 2—4-bit memory address register with two bit failures

(i) Clear memory.
(ii) Write into location (15) some pattern X. The pattern we have written will not be written into (15), but into (14).
(iii) Read (see Table I).

Table I—Memory address register diagnosis

Location	Actual Location Accessed	(Location) = X?	Diagnosis
1110	1110	Yes	Bit 0 is bad
1101	1100	No	Bit 1 is okay
1011	1010	No	Bit 2 is okay
0111	1110	Yes	Bit 3 is bad

If (14) contains X, bit 0 is bad
(13) contains X, bit 1 is bad
(11) contains X, bit 2 is bad
(7) contains X, bit 3 is bad

Thus the diagnostic is able to detect and isolate multiple errors in the MAR. Any number of MAR bits that are stuck to "1" or "0" will be diagnosed.

2. *Circuit and Stack Diagnosis.* For diagnosis it is necessary to know which locations were found bad, on which (ones or zeros) test they failed, and how many bits and which bits within a character were bad. If one has to store exactly which locations were bad, prohibitively large buffers would be necessary. Thus a better system to store error locations has to be found.

The selection of a particular X-line and Y-line results in a particular location being accessed. For the selection of an X-line, an X-selection switch, an X-current generator, an X-driver output have to be selected (see Figure 6). In Figure 1 it is seen that a current generator drives 8 drivers—4 drivers in each stack.

Let us see what happens if X-driver No. 3 in stack

No. 1 is always high. X drivers are decoded from bits R15, R06, and R05. The above X-driver has at its input $\overline{\mathrm{R15}}$.R06.R05. In addition, the X-current generator with $\overline{\mathrm{R07}}$ at its input has to be selected. Thus, all locations in the area $\overline{\mathrm{R15}}$.R06.R05.$\overline{\mathrm{R07}}$ will appear bad on a ones test. These locations are the blocks 30 - 37 (see Figure 4.) In this case errors would have been found because no current would have been supplied to the X-line.

Now consider a case where the output of the same driver is always low. In this case, all locations corresponding to $\overline{\mathrm{R07}}$.$\overline{\mathrm{R15}}$.$\overline{\mathrm{R06}}$.$\overline{\mathrm{R05}}$, $\overline{\mathrm{R07}}$.$\overline{\mathrm{R15}}$.$\overline{\mathrm{R06}}$.R05. $\overline{\mathrm{R07}}$.$\overline{\mathrm{R15}}$.R06.R05. $\overline{\mathrm{R07}}$.R15.$\overline{\mathrm{R06}}$.$\overline{\mathrm{R05}}$. $\overline{\mathrm{R07}}$.R15.$\overline{\mathrm{R06}}$.-R05, $\overline{\mathrm{R07}}$.R15.R06.$\overline{\mathrm{R05}}$, $\overline{\mathrm{R07}}$.R15.R06.R05 will appear bad. This is because when these locations are accessed, a current split takes place, i.e., the current from the selection switch splits between two drivers—the one selected and X driver No. 3 in the first stack.

Thus, in this case blocks 00–07,10–17,20–27 in the first stack and 00–07,10–17,20–27,30–37 in the second stack will appear bad.

The next case to be considered is when an input diode opens up. Assume that the diode input for R05 is open at the input to X-driver No. 3. This X-driver will come up in the following cases:

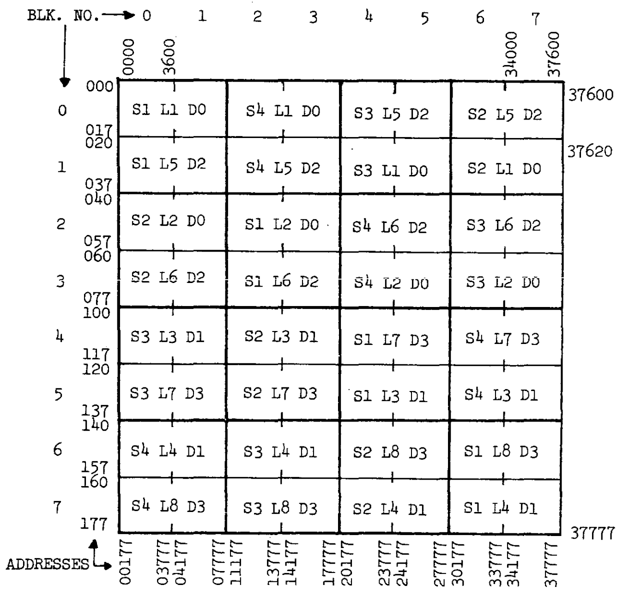
When locations with: $\overline{\mathrm{R07}}$.$\overline{\mathrm{R15}}$.R06.$\overline{\mathrm{R05}}$

$\overline{\mathrm{R07}}$.$\overline{\mathrm{R15}}$.R06.R05 in their addresses are accessed.

Every memory is driven by circuits that essentially form a decoding matrix where the decoding is done from the bits in the MAR. (See Table II).

Table II—Memory address register decoding
Bits numbered from least significant end R01-R18

Circuit	MAR Bits No.'s
X - Selection Switches	R01, R02, R03, R04
X - Read/Write Drivers	R05, R06, R15
X - Current Generators	R07
Y - Selection Switches	R08, R09, R10, R11
Y - Read/Write Drivers	R12, R13, R15
Y - Current Generators	R14
Stack Select	R15
Drawer Select	R16, R17, R18
Inhibit Driver-	R14, R07, R05
Inhibit Gates	R15, R06



S = SENSE LINE NO.
L = INHIBIT LINE NO.
D = INHIBIT DR. NO.

Figure 3—Functional map showing relationship between addresses and sense, inhibit lines

In addition there is the property of uniqueness, i.e., for every combination of bits in the MAR only one location is accessed. Thus the memory can be represented by a functional map in which the variables are the bits in the MAR.

Such a map has been drawn for the H-1200, H-2200 memory plane (see in Figures 3, 4). For purposes of convenience, the plane has been sectioned into 64 blocks (numbered from 00 to 77) of 16 × 16 locations each. Now the problem of determining error patterns becomes straightforward. Except in a few cases, all error patterns can be classified as distinct sets of blocks (see Figure 5). A few examples will clarify this. Of these two cases, the second is legitimate. Therefore, blocks 10-17 in the first stack will appear bad.

This leads us to the problem of resolution. If it is found that blocks 30-37 in the first stack are bad and all other locations are okay, we can conclude that the failure is due to one of the following possibilities:

a. X driver No. 3 in stack No. 1 may have failed.
b. The trouble may be due to an input diode (see Table III).

Table III—Resolution problem
Blocks 30–37 in Stack 1 Bad

CIRCUIT	INPUT	BAD DIODE ON CIRCUIT
X Dr. #3 Stack 2	R15.$\overline{\text{R07}}$.R06.R05	R15
X Dr. #1 Stack 1	$\overline{\text{R15}}$.$\overline{\text{R07}}$.$\overline{\text{R06}}$.R05	$\overline{\text{R06}}$
X Dr. #2 Stack 1	$\overline{\text{R15}}$.$\overline{\text{R07}}$.R06.$\overline{\text{R05}}$	$\overline{\text{R05}}$

AMNESA can guess whether the errors are due to case (a) or case (b). It is found that in case (b) all the locations in blocks 30-37 are bad. Even though there is a current split, some locations seem to work. In case (a) all locations in blocks 30-37 look bad.

Even though AMNESA does indicate the probable cause of failure in this case, it also indicates all other possibilities.

Failures of other types of circuits are diagnosed in a similar way. From Figures 3, 4 and Table II it can be seen that knowing the bits of the MAR which feed the input gating to the circuit, one can predict the error pattern that will be found when a circuit failure occurs.

It can be seen that some patterns will be subsets of larger error patterns. For example, when a current generator fails, the error pattern is a superimposition of the error patterns for 8 drivers—4 in each stack (see Figure 5). AMNESA does not print anything conclusive until all possible patterns are examined. It can also be seen that multiple failures will not confuse the diagnostic because multiple failures will only result in superimposed error patterns. As soon as AMNESA finds the largest error pattern, it diagnoses the failure, prints it out and halts. Other failures can be diagnosed by fixing the problems one at a time and rerunning AMNESA.

Failures in windings result in unique error patterns which are easy to recognize. The locations associated with each sense line and inhibit line are indicated in Figure 3.

Failures in bit circuitry are also diagnosed by AMNESA. Inhibit driver and sense amplifier failures lead to error patterns that are superimposed patterns for inhibit or sense lines in both stacks. The decoding for inhibit drivers is done on three decode packages. Failures in the decode packages will result in error patterns corresponding to failures in inhibit drivers. However, the outputs from the inhibit decodes feed

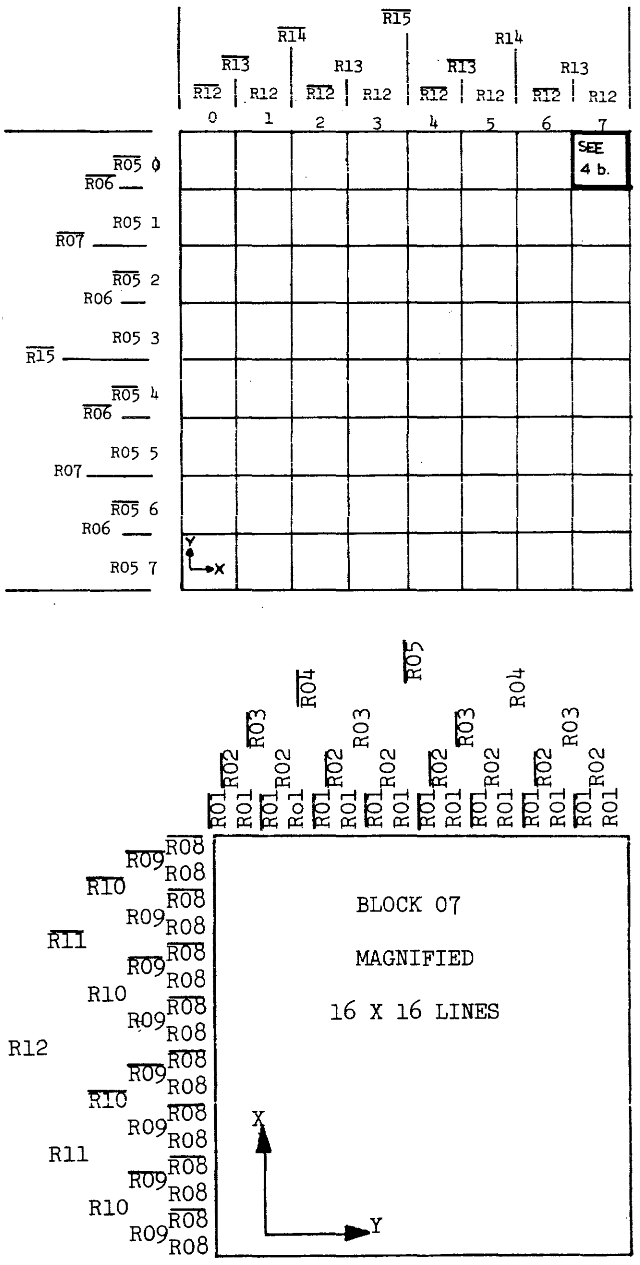


Figure 4—(a) Functional map showing how decoding structure relates to addresses (b) Block 07 in (a) enlarged

inhibit drivers in all nine planes of each stack. Thus, it is easy to distinguish between inhibit decode failures and inhibit failures even though the error patterns are the same. In the former case all bits of a character are affected while in the latter case only one bit is affected.

Parity plane testing is outlined in Figure 7. When errors are found by AMNESA and no error pattern is apparent, i.e., the errors look random, the diagnostic assumes that these are due to bad cores. A

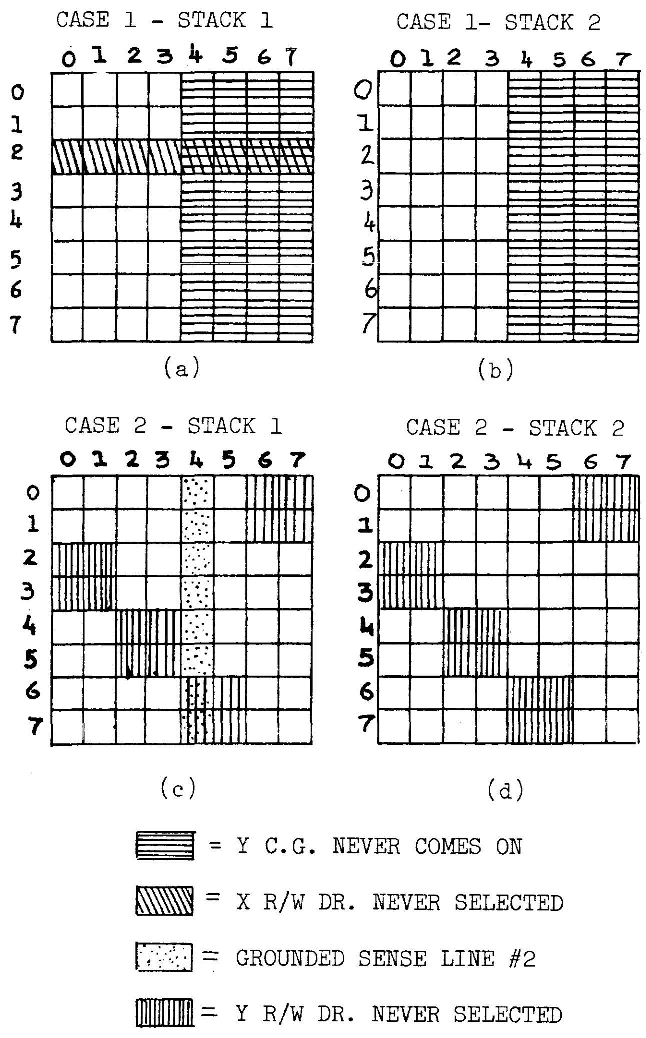


Figure 5—Examples of error patterns. In both cases multiple errors are present

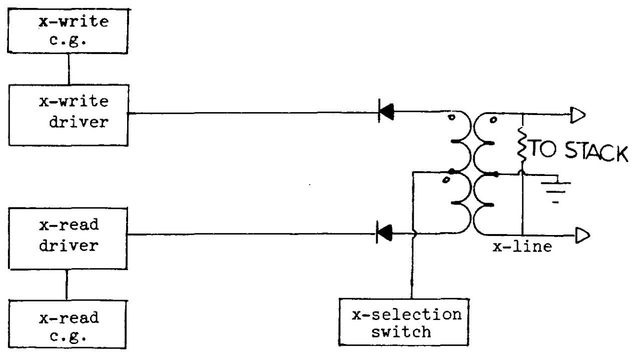


Figure 6—Diagram showing drive line selection scheme

printout of bad error locations is provided. Examples of error patterns are given in Figure 5.

The approach taken has been to represent the

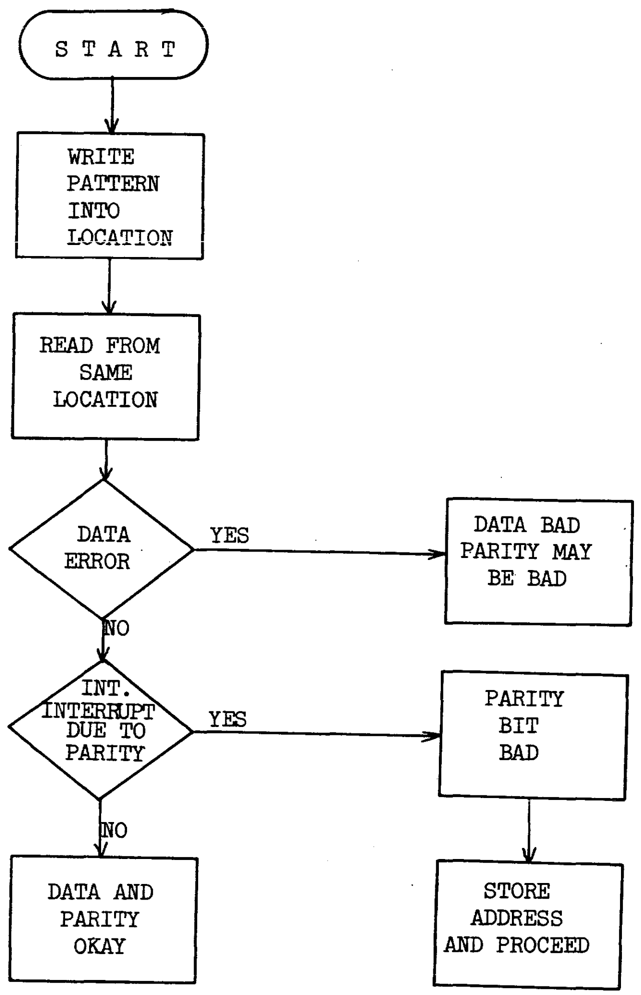

START
WRITE PATTERN INTO LOCATION
READ FROM SAME LOCATION
DATA ERROR
YES
NO
DATA BAD PARITY MAY BE BAD
INT. INTERRUPT DUE TO PARITY
YES
NO
PARITY BIT BAD
DATA AND PARITY OKAY
STORE ADDRESS AND PROCEED

Figure 7—Parity plane testing

memory functionally and to choose a convenient block size so that most error patterns can be expressed in terms of blocks. In most cases it is enough to know which block a particular faculty location belongs to and the exact address does not have to be stored. AMNESA initially just counts the number of errors per block and then proceeds to correlate this information with expected error patterns. Memories can generally be subdivided into blocks so that such an approach is always possible. In addition to simplifying programming, such a functional map helps to visualize error patterns (see Figure 5).

AMNESA

1. *General.* The program itself is composed of two segments. The first segment tests the memory drawer with different tests and determines and stores error patterns found. This is followed by preliminary diagnostic routines that search for patterns that would

be caused by failures of circuits in the "0" level (see Table IV).

The second segment is composed of diagnostic routines that try to correlate the failures detected by the first segment. This part of AMNESA searches for failures in circuits in higher levels.

2. *Printouts.* The first segment of each program has four possible printouts for each circuit tested. They are:

a. PACKAGE FAULTY*****
b. NO ERROR FOUND
c. PROBLEM SEEMS TO BE ELSEWHERE
d. TEST INCONCLUSIVE*

The second segment can output:

e. READ WHAT THE DIAGNOSTIC HAS PRINTED OUT PREVIOUSLY, or
f. DISREGARD PREVIOUS PRINTOUTS

X X X X X X X X X X X X

The X's above indicate an error message. An example of a printout with actual errors is shown in Figure 8.

In the case of intermittent failures in any circuit, the second segment will print:

READ WHAT THE DIAGNOSTIC
HAS PRINTED PREVIOUSLY

The operator then scans the first segment printouts for "TEST INCONCLUSIVE." If a test for any circuit has resulted in an inconclusive test *and* no other failures exist, that circuit has a malfunction.

3. *Resolution.* Given that an error has occurred, the fault causing the error is usually indistinguishable between X packages. This X is referred to as the resolution of the diagnostic.

```
------------------------------------------------------------------
DISREGARD PREVIOLS PRINTOUTS

ERROR IN PARITY PLANE

INHIBIT URIVER NO. 0 SEEMS BAC - SEE COMMENT 7
------------------------------------------------------------------

    COMMENT 7  **************************
    CNE OF SEVERAL THINGS MAY BE WRONG.

    A.  GC BACK ANC LOOK AT THE RESULTS FUR THE 0'S TEST FOR
        THE INHIBIT CRIVERS.

        IF YCU HAVE A BIT ERROR AND IF UNE CR MORE OF THE INHIBIT
        DRIVERS HAVE BEEN INCICATED FAULTY CR INCONCLUSIVE, CHECK
        THE PACKAGE CONTAINING THE INHIBIT DRIVERS FOR THAT BIT.
    IF ERROR PERSISTS, CHECK THE THE TWU SENSE PACKAGES FOR THAT BIT.

    IN THE CASE OF A WORD ERRCR -

    B.  CHECK THE INHIBIT CECCCE PACKAGES.
```

Figure 8—Printout with inhibit driver faulty

What one would ideally like is to have diagnostic with a resolution of 1 package for all possible errors. Unfortunately, this is not always possible.

For all transistor failures the resolution of AMNESA is 1. For diode failures the resolution varies from 1 to 8 in the worst case.

The above is not as bad as it looks because the ratio of transistor failures to diode failures is about 5 to 1. The chances of cases occurring in which the resolution of AMNESA is 8, are relatively very low.

The pity is, AMNESA isolates the fault down to 4 to 8 particular diodes in the worst case. Unfortunately, these diodes can be on different packages—resulting in a low resolution.

4. *Limitations.*

a. In many cases, shorted diodes will most probably not be found by AMNESA. Fortunately, the probability of such failures is relatively very low.
b. Open sense lines will lead to noise on the sense lines. The results are also unpredictable.

SUMMARY

The approach that has been used in the writing of the diagnostic is applicable to any memory system as it does not concern itself with the type of device used as much as with the memory organization. In general, the selection of a location in memory is done by a decoding matrix and the failures of circuits in the matrix should result in unique patterns in most cases. Representing the memory functionally allows us a convenient way of classifying patterns.

The failure of a circuit in a higher level will produce an error pattern that is a superposition of error patterns due to failures in lower levels. Failures in addressing circuitry will result in errors in all bits of a word, while failures in bit circuitry will result in errors in one bit (or a few bits in the case of multiple failures in bit circuitry) of a word.

If the memory devices themselves are not functioning correctly, the errors produced will be random in all probability. Thus, if the diagnostic determines that circuits associated with the memory have not failed, chances are that the errors are due to the memory devices themselves, or due to lines associated with only those devices. This in itself is a successful diagnosis.

Multiple failures are really no problem because the diagnostic does not attempt to correlate the total error pattern it discovers. The diagnostic rather decomposes the total pattern into its elements.

If all memory appears bad, then the diagnostic cannot even hope to be successful because this would appear to the diagnostic as though every circuit associated with memory has failed. However, there are not too many conditions that can cause all of memory to appear bad. These conditions can be given to the operator. Once he gets a part of the module working, the diagnostic can take over. The algorithm for the MAR can also be used in any memory system as the algorithm only capitalizes on the uniqueness of memory addressing.

The problem of improving resolution has some ramifications that apply to packaging during the design of a memory system.

AMNESA has been tested out by introducing actual faults into the memory system and has proved to be quite effective. The program has now been released to the field and preliminary reports indicate it is quite successful. The same approach is being used to write memory diagnostics for the H-200 (coincident current memory) and the H-4200 and the H-8200, which have 2 ½ D memories.

Table IV—Classification of circuits

	LEVEL	NUMBER	COMMON
(i) X-current generators	2	4/D	Yes
(ii) Y-current generators	2	4/D	Yes
(iii) X R/W Drivers	1	8/S	No
(iv) Y R/W Drivers	1	8/S	No
(v) X-selection switches	1	16/D	Yes
(vi) Y-selection switches	1	16/D	Yes
(vii) Sense amps, sense flops, etc.	0	--	Yes
(viii) Inhibit drivers	0	4/P	Yes
(ix) Inhibit gates	1	2/S	No
(x) Inhibit decode packages	2	3/D	Yes
(xi) Memory Address Register	3	--	Yes
(xii) X-diodes	0	--	No
(xiii) Y-diodes	0	--	No
(xiv) X-transformers	0	--	No
(xv) Y-transformers	0	--	No

The circuits are classified according to levels for purposes of explanation. For example, X-current generators drive X-drivers. X-current generators are, therefore, in a level higher than X-drivers, i.e., if an X-current generator never comes on, it looks as though 4 X-drivers in each stack have failed.

The column under "Number" indicates how many circuits there are per plane stack or drawer.

D = Drawer

S = Stack

P = Plane

The column under "Common" indicates whether the circuits are common to two stacks.

Characteristics of faults in MOS arrays

by H. R. LAMBERT

North American Rockwell Corporation
Anaheim, California

INTRODUCTION

Before discussing characteristics of faults in MOS arrays, it appears desirable to review briefly MOS technology, and to present some discussion on the use of MOS devices in arrays. Not only will this benefit those not too familiar with MOS devices, but it will also serve as a basis for the subsequent discussion on failure modes and mechanisms.

Metal-Oxide-Semiconductor (MOS) device fabrication employs the same basic technologies used in the fabrication of bipolar devices. However, MOS devices are quite different from bipolar devices in operation. To understand some of the problems that may be encountered in MOS devices, it is thus necessary to know something of their construction and operation.

Figure 1 shows a schematic cross-section of the type of MOS field effect transistor (FET) that is the basic element for most MOS arrays. This figure also shows the symbolic representation for a MOSFET. This type of MOSFET, called a P Channel Enhancement MOSFET, is made by diffusing two P regions into an N type silicon substrate. A metal film control element, called a gate, is evaporated onto an insulating film over the region separating the two P regions. It is from this feature that the term Metal-Oxide-Semiconductor is derived. This can be somewhat of a misnomer, however, as the insulating layer between the gate and the substrate need not be silicon oxide. It may be silicon nitride, or a silicon oxide/nitride combination. This latter type of device may be called an MNOS device, for Metal-Nitride-Oxide-Semiconductor.

If the Source P region of a P Channel device is grounded with the substrate, as shown in Figure 2, and negative voltage is applied to the Drain P region, there will be no appreciable drain to source conduction as long as the gate is unbiased. This off state has the megohm impedance of a reverse biased P–N junction. When negative voltage is applied to the gate, minority (positive) carriers are attracted to the surface of the substrate that is covered by the gate. At the same time, majority (negative) carriers are repelled from the surface. When sufficient negative gate voltage has been applied, the N type material under the gate will "invert" and a P type path or channel is formed between the drain and source. The gate voltage at which this occurs is called the threshold voltage and is generally defined as the gate voltage required to allow a specified drain to source current at a specified drain to source voltage. As the gate voltage is made still more negative,

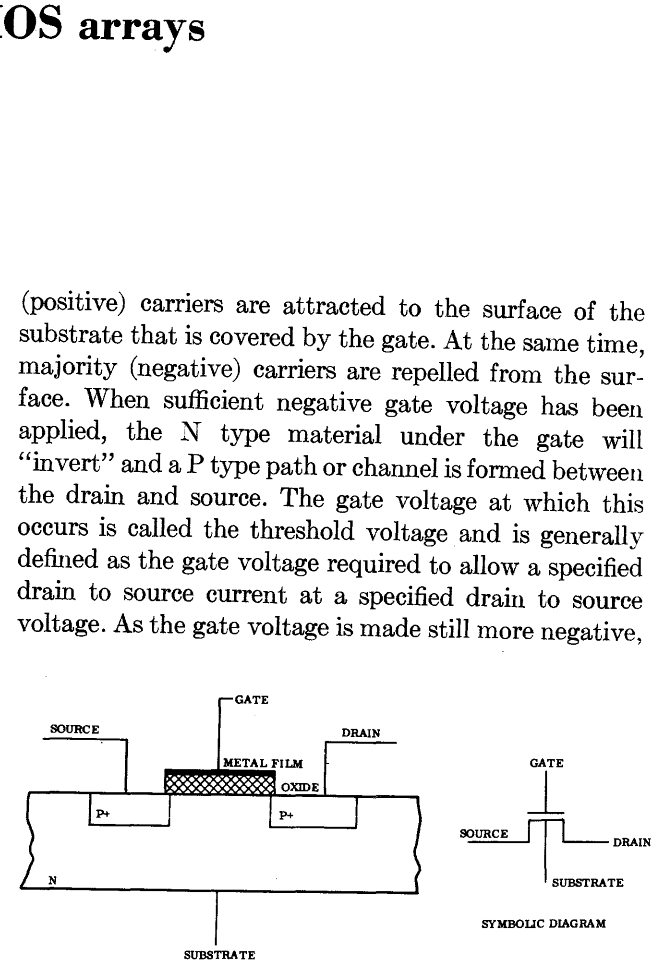


Figure 1—Cross section of P channel enhancement mode MOSFET

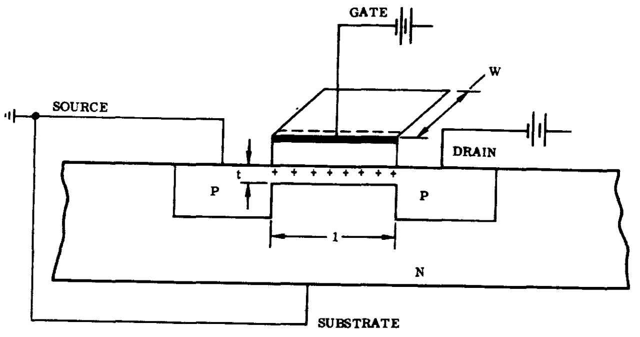


Figure 2—Biasing and operation of MOSFET

the induced channel is deepened and the drain to source conductivity is increased. This increase of conductivity by increase of gate voltage is referred to as enhancement mode operation. When the gate voltage level is subsequently reduced below the threshold voltage, the channel reverts to its initial N type and the device no longer conducts. The control of conductivity by an electric field gives rise to the descriptive name of field effect transistor. Because gate is separated from all other portions of the device by an insulating layer, there is essentially no gate current required for operation.

It is also possible to fabricate N channel devices by diffusing N regions into P type substrates and to make devices with a thin diffused channel between the drain and source. This latter type results in a normally on device in which the drain to source conduction can be either increased (enhanced) or decreased (depleted). Such a device is usually called a depletion type device to distinguish it from the above described enchancement mode devices. Junction type FETs are also fabricated, but will not be discussed because of their current unimportance in integrated circuitry.

Figure 3 shows what are commonly called drain characteristics, in order to present some idea of the electrical characteristics of MOS devices. This figure plots the variation of drain-to-source current with gate and drain voltages for a typical P channel enhancement device. Other typical electrical characteristics are threshold voltages of 3–5 volts and on resistances of on the order of 100 ohms.

More should be said about on resistances at this point, as this is one of the characteristics of MOSFETs which makes them particularly applicable to integration into arrays. The drain to source on resistance is determined by the common resistance formula:

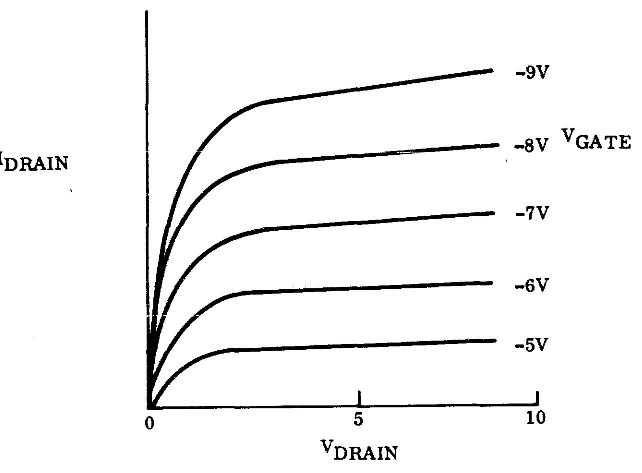


Figure 3—Drain characteristics of MOSFETS

$$R = \rho \frac{L}{Wxt} \tag{1}$$

L and W are the channel length and width; t is the thickness and ρ is the resistivity of the induced channel. A MOSFET fabricated with a long narrow channel has the characteristics of a load resistor when turned on. This technique is utilized in the fabrication of MOS integrated circuits to eliminate the area consuming diffused resistors required on bipolar integrated circuits.

The switch like characteristics of MOSFETs makes them ideally suited for digital applications. Because of low gate current requirements, MOSFETs can be made very small. Good isolation between elements allows dense packing of MOSFETs in integrated circuits. These features result in the ability to fabricate very complex MOS digital arrays on very small dice. As an example of the degree of complexity possible, Autonetics has fabricated 1024 bit shift registers containing over 6000 MOSFETs on 160 by 135 mil dice. Table I lists some of the type of MOS arrays currently available from semiconductor manufacturers so that the impact of MOS on the computer industry can better be appreciated.

Although generalizations are difficult to make because of the many and varied types of MOS arrays available, for the purposes of this discussion an attempt will be made to categorize at least the logic arrays into static and dynamic types. The former use flip-flop type elements to store information as long as desired and are capable of d-c operation. Dynamic arrays, on the other hand, utilize the high input impedance characteristics of gates (10^{14} to 10^{16} ohms) to store information for brief periods of time. As an example, consider what happens when the voltage is removed from the gate of a turned on device. If the voltage is removed by grounding, the gate which acts like a capacitor is discharged and the device will be turned off. If, however, the voltage is removed by opening the gate line (as by turning off a series MOSFET) the gate will stay charged until the charge leaks off. As long as the gate stays charged, the device will remain turned on. In this manner it is possible to store information for brief periods while other operations are being performed.

This technique is illustrated in Figure 4 which shows one bit of a dynamic, multibit shift register. When an input zero is applied to Q1 coincident with clock pulse 1 (CP_1), the gate of Q5 will be charged to $-18V$ from V_{DD} (through Q2 and Q3). Q5 will stay turned on after Q_2 and Q_3 are turned off by CP_1 returning to ground. When CP_2 turns on Q_4 and Q_6 any charge on Q_7 will discharge

Table I—Typical types of MOS circuit currently available

1—Shift Registers
- Static and dynamic registers ranging up to Quad 256 bit registers.

2—Memories
- Read only memories, nondestructive, to 1024 bits.
- Random access memories to 32 bits.

3—Converters
- D to A converters.
- A to D converters.

4—Counters
- Binary up-down counters.
- Variable modulus counters.
- Binary, ripple carry type counters.

5—Flip-Flops
- Dual J-K flip-flops.
- RST flip-flops.

6—Gates
- Various types of Dual and Quad multi input NAND/NOR, AND, and OR/NOT gates.

7—Miscellaneous
- Parallel accumulator.
- Full adder.
- Parity detector.
- Level shifter.
- Counter timer.
- Clock generator.
- Address decoder.
- Serial/parallel—parallel/serial converter.
- Other specialized types of arrays.
- Numeric character generator.
- Arithmetic comparator.
- Servo Adder.
- Frequency divider.
- Decimal point display.
- Digital differential amplifier.
- Core memory interface.

to ground through Q_5, and a zero will have been propagated from the input of Q_1 to the input of Q_7 by the subsequent application of CP_1 and CP_2 pulses.

Although input gates of MOS arrays are generally protected from static electric discharges by low voltage diodes, the high input impedance of internal devices if not affected. The lower frequency limit of operation of dynamic arrays depends on the RC time constant of the internal gates and is on the order of 1 kHz.

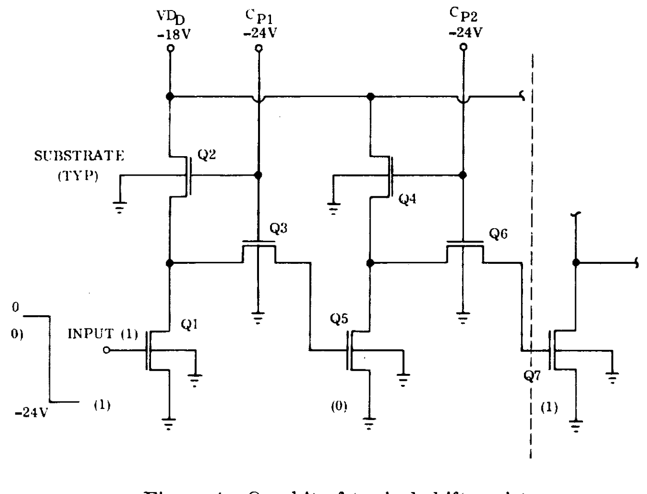


Figure 4—One bit of typical shift register

Discussion of fault characteristics

It was previously noted that the same basic methods are used for fabricating both MOS and bipolar devices. These include crystals growing, wafer preparation, oxide formation, masking and etching, diffusion, die and wire bonding, and packaging. The materials used for the substrates and doping are also similar. It would be correctly expected that processing and material related failure mechanism on both types of devices would be similar. The justification for discussion of MOS fault characteristics, when so much has been written on those of bipolar devices, is based on degree and manner that MOS devices are affected. These different degrees and manners are a result of the field method of operation of MOS devices and the necessary dimensional differences required by MOS devices for this type of operation. Discussion will be confined to fault characteristics related to these differences although tabular listings of all normally encountered failure mechanisms will be presented. An attempt will later be made to relate these discussed mechanisms to various types of MOS array malfunctions.

The discussion will confine itself to the types of faults that cause failure in services; that is, mechanism that cause apparently good arrays to malfunction at some subsequent time after they have been put into operation. These malfunctions, neglecting misapplication and overstresses, relate back to fabrication anomalies or design deficiencies, as such, they also relate to mechanisms responsible for yield loss during fabrication.

For the purpose of organization, the faults will be grouped into the following three categories.

1. Catastrophic faults that result in permanent complete device malfunction. Examples are opens for shorts caused by one or more of the conditions shown in Table II.
2. Intermittent faults that cause temporary device malfunction or that cause loss of any stored information. Examples are intermittent opens or shorts caused by one or more of the conditions shown in Table III.
3. Degradation faults that gradually cause complete device malfunction or malfunction under some conditions of operation. An example is leakage current degradation which could be caused by one or more of the conditions shown in Table IV.

Table II—Catastrophic failure mechanisms

Failure Indicator	Related Failure Mechanism		
Opens	1—Bonding	a.	Poor wire bonding (weak, off pads, etc.).
		b.	Intermetallics at pads.
		c.	Damaged wires.
		d.	Mechanical overstress (shock, etc.).
	2—Metallization	a.	Unopened contact windows.
		b.	Breaks in the metallization
		c.	Handling damage to metallization.
		d.	Metal migration.
	3—Mechanical Damage	a.	Cracked or broken die.
	4—Shorting	a.	Excessive current drain at short and elements subsequently burned open.
Shorts	1—Oxide defects	a.	Metallization penetrates oxide.
	2—Metallization	a.	Metallization lines touching as a result of mask defects.
	3—Bonding	a.	Bonding wires touching.
	4—Mechanical Damage	a.	Cracked die.
	5—Electrical Over Stress	a.	Oxide rupture.
		b.	Excessive power drawn causing melting of elements.
	6—Foreign Material	a.	Bridging between elements.

Table III—Intermittent catastrophic failure mechanisms

Failure Indicator	Failure Mechanism		
Opens	1—Bonding	a.	Loose bond, die
	2—Mechanical Damage	a.	Cracked die.
Shorts	1—Bonding	a.	Wires touching each other or die.
	2—Mechanical Damage	a.	Cracked die.
	3—Foreign Material	a.	Loose foreign material shorting between elements.

Table IV—Degrading failure mechanisms

Failure Indicator	Failure Mechanism	
High Leakage Currents	Contamination	a. In the oxide. b. On the oxide. c. On the package.
Shifted Threshold Voltage	Contamination	a. In the oxide. b. At the oxide—silicon interface.

The failure mechanisms listed in Tables II, III and IV are common, except to degree and effect, to both MOS and bipolar devices. Some mechanisms are dependent upon die and/or diffused region areas, others are dependent on the number of internal wire bonds used, still others are dependent on packing densities of elements and spacing of metallization lines. Some mechanisms are also related to the particular structure and operation of MOS devices. Table V illustrates these various dependencies.

As an example of what is meant by mechanisms being dependent on various geometric factors, consider the effect of die area and diffused (active) area on devices with oxide defects. The die area obviously does not cause oxide defects; however, if there exists a certain density of oxide defects per square cm, then statistically a large die is more likely to have defects on it than a small die. Similarly, a die with closely packed elements or large diffusion areas is more apt to have oxide defects in critical locations than the same size die with a lower packing density and smaller diffused areas. These geometrical considerations indicate that it is reasonable to expect a large, complex array to have a higher failure rate than a small, simple array. This would normally be offset by system requirements for fewer of the more complex arrays, with the resultant fewer interconnect and mounting problems. At any rate, geometrical effects are not peculiar to MOS devices and will not be discussed further.

Table V indicates that oxide defects, surface contamination, and metal migration are all influenced by the MOS structure and method of operation. Experience has also indicated that these three mechanisms are the largest causes of MOS failures under stress. These mechanisms and their relation to MOS devices will be discussed below.

Oxide defects can be holes, cracks, or weak spots in the oxide insulating layer. If these defects occur under

Table V—Failure mechanisms and dependency factors

Failure Mechanism	Dependent Factors					
	Geometry				MOS	
	Die Area	Diffusion Areas	Packing Density	No. of Bonds	MOS Structure	MOS Operation
Bond discrepancies				X		
Metallization defects			X			
Metallization migration			X			X
Cracked or broken dice	X					
Oxide defects	X	X	X		X	
Foreign material			X			
Surface contamination	X					X

interconnect lines whose metallization subsequently penetrates the defect, the interconnect line becomes shorted to the underlying element or substrate. Most oxide defects appear to be caused by surface irregularities present on the silicon prior to oxidation.[1] Assuming a uniform distribution of silicon surface irregularities, more oxide defects will occur in regions of thin oxide and at abrupt oxide steps than in uniform regions of thick oxide. The threshold voltage of MOS devices is governed by the following relationships:[2]

$$V_T = (Q_{SS} + Q_{BB}) \frac{t_{ox}}{e_{ox}} \tag{2}$$

where t_{ox} is the oxide thickness, e_{ox} is the dielectric constant of the oxide, and Q_{SS} and Q_{BB} are the charge densities at the silicon-silicon oxide interface and in the substrate respectively. In order to obtain thresholds in the usable range of 3 to 6 volts, it is necessary to have a gate oxide thickness on the order of 1500A°. In comparison, the thinnest oxide normally found on bipolar devices is in the 5000A° to 10,000A° range. There is also an oxide step around the periphery of the gate oxide where the oxide thickens to several thousand Angstroms over the P regions. Inasmuch as the gate oxide and surrounding step are covered by metallization any oxide defects in these regions are potential shorts between the gate metallization and underlying substrate and P regions. The effect of such shorts will be discussed later in the paper.

Contamination in and on the insulating layer or at the silicon-silicon oxide interface can result from impurities in the various cleaning and etching solutions or volatile impurities in the oxidation and diffusion furnaces. The various exposures to high temperatures during fabrication tend to uniformly distribute the contamination ions and the effect of their presence is usually not detectable by just electrically testing the completed device. Small amounts of contamination can, however, result in considerable device performance degradation in subsequent use.

The operation of MOSFETs by application of an electric field at the gate was discussed in the introduction. The formation of a conducting drain to source channel by inversion of the substrate surface under the gate was described. This inversion was seen to be caused by attraction of minority carriers and repulsion of majority carriers by applied gate voltage. The gate voltage required to initiate drain to source conduction across the inverted region was defined as the threshold voltage.

Any spurious charge concentrations under the gate affect Q_{SS} (reference equation 2), thereby, inhibiting or enhancing the channel formation process and shifting the threshold voltage. If gate bias is applied at ambient temperatures above 100°C mobile contamination ions become concentrated at the substrate surface as shown in Figure 5. This figure shows the tendency of positive contaminant ions to increase threshold voltages in P channel MOS devices by inhibiting the inversion process. It appears that the spurious positive ions result from contamination by sodium present in the various fabrication materials and equipment. Considerable effort is normally expended to reduce the level of sodium contamination to an acceptable level.

In addition to inhibiting channel formation, positive ions can cause surface inversion of P regions by attracting negative minority carriers and repelling positive majority carriers. In most types of MOSFET's there is some gate metallization overlap of the drain P region. If at high temperatures the drain is biased negatively in respect to the gate, positive charges will accumulate in the gate overlap region as shown in Figure 6. This will cause a distortion of the drain to substrate junction in this region giving rise to an increase drain leakage current when the device is in the off state.

The effects of increased threshold voltages and leakage currents will be discussed later.

Metal migration occurs when metal ions are knocked

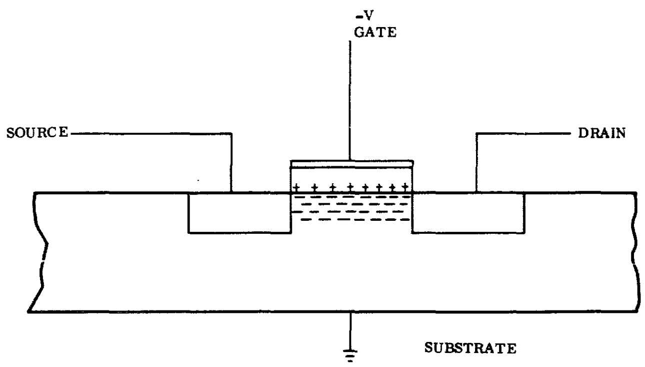


Figure 5—Accumulation of channel surface

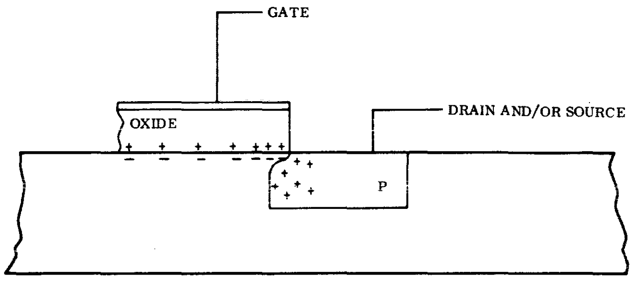


Figure 6—P region inversion

out of their crystal lattice by high velocity conduction electrons. The migration rate is accelerated by high temperatures and high current densities and is influenced by the crystal structure of the metal. Considerable investigation has been done on the migration of aluminum films such as are used for interconnects in integrated circuits.[3,4,5] The small cross sectional area of these interconnects can result in current densities of 10^5 to 10^6 amps/sq cm for only a few milliamps of current. By way of comparison, this current density is about two orders of magnitude higher than that which occurs in high power transformers. As elements become more densely packed in complex arrays, the interconnect lines must be made narrower. Care must be taken to limit current densities within safe limits by application of appropriate design rules.

The foregoing applies to any arrays employing aluminum interconnects, but a special problem exists on MOS arrays in lines used to charge large numbers of gates. Normally gate lines are considered to draw very little current. However, gate capacitance charging by fast rise time clocks gives rise to large instantaneous currents with resulting instantaneous current densities of 10^7 to 10^8 amps/sq cm. Although no significant relations have been obtained between these high instantaneous current densities and migration rates, conservative design rules must be employed to anticipate the potential problem.

Previously it was noted that the three failure mechanisms most influenced by MOS structure and operation are the same ones that are responsible for most MOS array malfunctions under stress or use. These mechanisms result in gate shorts, threshold and leakage increases and open interconnections. The question to be examined is how such discrepancies can be expected to affect operation of typical digital circuits.

The function of all types of digital arrays is to store, read out or perform some operation on digital information at the command of some input. The resultant digital outputs are in the form of "ones" and "zeros." When an array malfunctions it can be expected that the outputs will be in error in one of the following ways:

1. Outputs are all "ones," all "zeros" or some intermediate level for any input or operation.
2. Outputs that should be "ones" are "zeros" and vice versa, either permanently or intermittently.
3. Outputs are correct but are degraded to the extent that other arrays will not correctly recognize them.

In some cases these malfunctions may be temperature, voltage or frequency dependent.

It can be expected that gate shorts and interconnect opens will cause permanent (as opposed to intermittent) arrays malfunctions such as 1 or 2 above. These malfunctions will in addition be quite insensitive to temperature, voltage or frequency variations about their nominal values. The location at which the discrepancy occurs will govern the type of malfunction. A short in one place may cause an all "zero" output while a short at another place may cause an all "one" output. In addition a short at one location may produce the same effect as an open interconnect at a different location. Certain types of circuits may have weak areas in fixed locations so that devices tend to fail in the same manner, causing repetitions of the same circuit malfunction.

If shorts are responsible for the array malfunction, loading down of clocks, inputs or suppliers may occur. This loading may be to the extent that other good arrays fed from these sources may not function properly. Open interconnects will generally not cause such loading down, and in fact may reduce the normal load.

Arrays with degraded threshold voltages or leakage currents may exhibit any of the three above listed malfunctions at nominal temperature, voltages and frequency. However, these malfunctions will generally be sensitive to variations of temperature, voltages and frequency about the nominal values, as will later be discussed.

It was shown in Figure 3 that for a given drain-to-source voltage, increasing the gate voltage above its threshold results in an increase of drain to source current. That is, the channel resistance is reduced. For a constant gate voltage, as the threshold voltage increases, the gate is turned on less hard and the channel resistance increases. Thus, any gate charging and discharging through another device whose on resistance has increased as a result of threshold degradation will charge and discharge at a slower rate. The device will therefore turn on and off more slowly. If a number of threshold degraded devices are employed in series, as for example in a multibit register, the cumulative effect may impair high frequency operation of the array. Low frequency operation in which the devices are allowed enough time for this slower turn on and off will not be affected. At intermediate frequencies intermittent operation and incomplete transitions between the "one" and "zero" levels may occur. Devices affected by degraded thresholds will not be greatly sensitive to temperature or voltage variations.

As shown in Figure 4 devices such as Q3 and Q6 are used to control the gate charging of other MOSFETs. When the controlling devices are turned off, any charge present on the gates of MOSFETs they control will be stored until it leaks off. This means that the controlled device will stay turned on as long as its gate stays charged. Information may be stored in this manner. The length of time a gate will maintain its stored charge under such type of operation determines the lower cutoff frequency of operation. If there is an enhanced charge leakage path through the control device as a result of P region inversion, the gate being controlled will loose charge faster than intended and low frequency operation of the device and array will be impaired. At higher frequencies charges are required to be stored for shorter durations and array operation may be normal. Low frequency operation may be improved by increasing the voltages to the array as the gates are then charged to higher levels and a longer time is required for them to discharge. Operation at high frequency may be impaired by elevated ambient temperature which increase leakage currents. At frequencies, voltages and temperatures intermediate between those causing normal operation and inoperation, outputs may become degraded so that "ones" appear only as sharp spikes. Such degraded "ones" may not be recognized by other arrays into which they feed.

The leakage currents on an array may degrade to the extent that considerable loading down of signals occur. In addition enough power may be consumed by the array to significantly increase its temperature. As leakage current increases with temperature a thermal run-away condition may occur with the resultant destruction of the device. Even if thermal runaway is not encountered, the array temperature may increase enough to further impair low frequency operation.

It is characteristic of contamination associated failure mechanisms that the degradation introduced over a period of time in use can be baked out at 200°C in only a few hours time.

It should be remarked that some oxide defects may cause high resistance leakage paths which will have the same affect on device performance as high leakage currents resulting from contamination. It is expected that such type failures would be less temperature sensitive, and could not be returned to normal operation by baking.

SUMMARY

The fabrication techniques and mode of operation cause oxide defects, contamination, and metal migration to be the dominant failure mechanisms of MOS devices.

Shorts caused by oxide defects and opens caused by metal migration give rise to permanent voltage, frequency and temperature insensitive array malfunctions. Array outputs may be all "ones," all "zeros," or outputs that should be "ones" may be "zeros" and vice versa.

Contamination, notably positive sodium ions, cause thresholds and/or leakage currents to increase. High frequency operation is affected by the former and low frequency operation by the latter. The latter is also sensitive to high temperatures and low voltages. Outputs on contaminated devices may be similar to those for opens or shorts or may be degraded correct outputs.

REFERENCES

1 J E MEINHARD
Process techniques study of integrated circuits interim scientific
Report No C5-147/ 12/501
Prepared under NASA contract NAS 12-4 May 1968

2 P J BESSER J E MEINHARD P H EISENBERG
Factors influencing dielectric defects in silicon oxide layers
Presented at Electro Chemical Society Philadelphia Pa
October 1966 Autonetics Report # X6-3072/501

3 J R BLACK
Mass transport of aluminum by momentum exchange with conducting electrons
IEEE Sixth Annual Reliability Physics Symposium
Proceedings 1967 Vol. 6

4 I B BLECH H SELLO
The failure of thin aluminum current carrying strips on oxidized silicon
Physics of Failure in Electronics Vol 5 1966

5 R V PENNY
Current induced mass transport in aluminum
Journal Physics of Chemical Solids Vol 25 1964

A systematic approach to the development of system programs

by F. M. TRAPNELL

Qandac Associates Limited
Zug, Switzerland

INTRODUCTION

The brief history of the development of large programming systems shows a persistent inability to predict cost and time associated with it. For this reason I want to discuss some of the principles and practice which experience (either good or bad) has shown can yield a higher level of predictability. I do not pretend that these principles are always easy to apply or that they do not have to be interpreted to particular situations. However, I do believe that the principles are fundamental in that predictability can be guaranteed to increase if they are followed.

Rapidly changing system technology is causing rapid increases in the scale of functional complexity and sheer size of these systems which in turn has caused rapid increases in the cost of and in the time required to develop them. Thus, in any given undertaking there will continue to be very little precedent for the leaders in the technology to follow in developing successive systems. Given that there is little precedent to follow, it should come as no surprise that the main theme of this paper is to call for carefully detailed and explicit planning of all phases of the development project.

Specification

Of all the problems facing the developer the lack of an adequate systems specification is both the hardest to overcome and at the same time the one which, if not overcome, will cause the most grief. This specification is a statement of what the system will do under all conditions, how it will do it, how fast it will operate, and how large it will be. The choice of language in which the specification is written is important in that it must be understandable to a wide range of people: the programmers, the system engineers, the managers, the users, and so forth. This, combined with the need to be both explicit and detailed regarding all situations in which the program might find itself, makes it an extremely difficult document to produce. From a practical point of view, however, one can regard it as axiomatic that what is not contained in the specification will not be in the system when it is finally built.

Structure

An important factor in achieving an adequate specification is deciding on the structure of the specification, itself. Only a few people can have a thorough knowledge of a large system; yet it may be necessary to have a large number of people working on programming and testing various parts of the system in order to complete it on schedule. To accomplish this, the structure of an overall specification must be such that it can easily be broken into as many sub-specifications as are required. The process is similar to constructing a bill of materials explosion in planning a manufacturing process. The cleavage into sub-programs will depend upon considerations of modularity, how maintenance of the system is to be carried out, how important is it to have the ability to substitute alternate program modules, and the fact that there may be natural boundaries between functions of the system which make natural interfaces. In any event, and however this is done, the size of each specified module should be no bigger than that which one man can program and test during the allotted development cycle. Thus a module becomes the property of one individual, who must be held responsible for achieving its specification, including proper interfaces, proper use of system facilities, function, speed and size.

Cross system communications

With such a breakdown of specification, it becomes a major task to control the communication between these modules. Ideally a developer would like to ensure that these communications operate through system defined mechanisms. That is to say, every communication between modules would take place through macro instructions with symbolic parameters which are expanded into machine code and data either at assembly time or execute time or some of both. These expansions, and system tables, lists, rolls, or control blocks which they use, should be specified and controlled by the central specification control agency discussed earlier. Thus, they are part of the systems specification itself and they must not be changed by individual programmers.

One of the problems in achieving this ideal is that many of these high level communications facilities are, or seem to be, cumbersome and slow. Thus a strong case may be made for private communications between modules of the system where the two programmers in question understand in detail how their code works and therefore can use this internal knowledge to locally speed things up. I, myself, have succumbed to this kind of rationale when in the heat of development there did not seem to be any satisfactory alternatives. (This, incidentally, I attribute to not having done the kind of specification job that I described earlier.) On the other hand, I am satisfied that this internal communications approach is unsound in large systems and can in fact be disastrous if it is arrived at by private agreement without the overall knowledge of the specification control authority. The right way to deal with this performance problem is to define the communications problem accurately as part of the specification and then design the cross-system communication mechanisms so that they meet the performance required. If necessary the system developer should obtain additions or modifications to the hardware which help to solve these problems.

(Incidentally, one of the worst character defects of system programmers is that they will go beyond the point of reason to avoid demanding changes to hardware. Frequently this is due to their inability to discuss these problems intelligently with cost conscious engineers; but some part of it is due to the fact that they often regard it as their job to program around all hardware problems.)

One reason to control communications at the system level is to maintain knowledge of how communications ought to take place so that when system bugs occur they can be analysed more easily and systematically. A second reason is to provide the flexibility to change the communications paths, the communication media (e.g., Vector tables), as well as the location and identity of the source and destination modules without having to redesign and recode the system. This can pay dividends, not only as the system is changed and expanded but also during the testing phase when one can more easily put in 'scaffolding' programs to simulate the action of routines that have not yet been written.

Development of the specification

The development of an adequate specification is, of course, a continuing process. No one is smart enough to sit down cold to produce a completely adequate specification for a large, complex system any more than one could sit down cold to produce a completely adequate constitution to govern a state or nation. As the program develops, new knowledge and insight will be gained which will lead to the need to change the specification. On the other hand, the nearer in detail the initial specification is to the final one, the quicker the development and specification changing processes will converge to an adequate design.

I have rarely seen enough time and effort spent in achieving an adequate initial specification for a large system to optimise either the overall development time or the overall cost. Time can be wasted in over-designing certain phases of the system, and there is a strong temptation to feel that unresolved specification problems will somehow disappear in the course of implementation. Thus, an aggressive development manager may prematurely decide to stop designing and start programming. As the experienced high altitude aviator disciplines himself to recognize the insidious symptoms of the lack of sufficient oxygen, so the experienced developer should discipline himself to recognize the insidious symptoms of not having sufficient specification. In practice I would suggest the following:

The main inadequacy of specifications is that they are usually too narrow in scope. Typically, they cover in depth those parts of the system which the designers feel are elegant, sophisticated and new; often they either skim or avoid problems which the designers feel are not technically interesting. Hartman and Owens in their paper[1] in the proceedings of the 1967 FJCC indicate the sort of breadth required in a compiler specification. As they say, a special and frequently troublesome class of problems arises from a lack of consideration for user problems.

It is essential that the developer avoids becoming so engrossed in the detail of given aspects of the system that he does not achieve completeness in the breadth of conditions and situations for which the system is designed to operate. One way to achieve this is to arrive at a formal specification through an iterative process of successive specifications. At each stage:

a. some problems are solved and some design choices made; these in turn raise new problems
b. the designers attempt to be exhaustive in identifying and describing these outstanding problems to be solved in order to achieve an adequate design
c. they limit the extent to which they try to solve each problem.

Throughout this process the developers must concentrate on the question "Have we thought of all the situations and conditions which must be taken care of?" Until that quest is ended they must not be distracted by the fact that they may not have solutions in hand for all the problems encountered.

Figure 1 illustrates this process, using some suggested stages. The first stage is called External Requirements, which tells how the system appears to the outside world: what the users, operators, manager, etc., see. The second is a functional cleavage of the system: what functional block there will be, what each will do. The third is interfunction communication; this describes the linkages to programs, system control blocks, system tables, etc., in terms ot their symbolic names, linkage parameters required, data formats of parameter lists, etc.; at this point internal conventions and standards will be established and a control function for these will be appointed. In the fourth stage, called program module specification, the system is carved into modules to be programmed: input/output/function specifications are written for each module. These I/O/F specifications list every input (entry point), every output with its destination (exit point, interrupt, external reference, etc.), the linkage parameters and formats associated with each, the function which the module performs and its normal place of residence (in core, or disc, etc.). The fifth and final stage will provide detailed flow charts and final estimates of size and performance for each module.

To get the system design right these stages will have to be gone through more than once! For this reason the path on the left of Figure 1 shows a return to stage one at the completion of each stage. This is meant to imply that before proceeding to the next stage in sequence, the designers should go back through all previous stages to formally make the corrections found necessary as a result of the last stage completed.

These changes should be restricted to what is necessary to proceed; but on the other hand they give the designers the opportunity to correct mistakes made early in design. The programming manager who tries to short circuit the process will either end up with a poor system or will come face to face with the old system design adage which says: "There never seems to be

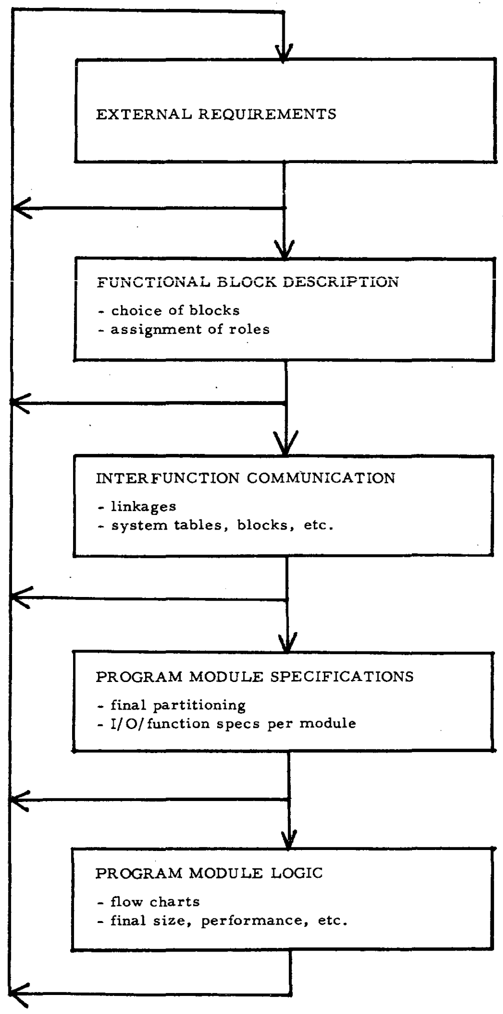

EXTERNAL REQUIREMENTS
FUNCTIONAL BLOCK DESCRIPTION
- choice of blocks
- assignment of roles
INTERFUNCTION COMMUNICATION
- linkages
- system tables, blocks, etc.
PROGRAM MODULE SPECIFICATIONS
- final partitioning
- I/O/function specs per module
PROGRAM MODULE LOGIC
- flow charts
- final size, performance, etc.

Figure 1—Specification sequence

time to do it right the first time, but there always turns out to be time to do it over again."

The overall objective is to achieve a specification on which to base confident extimates of feasibility and development cost and time. The minimum requirements for such a specification are:

For the overall system:

1. A summary description of system facilities, how the system works, and the intent behind system standards as well as linkage and access requirements for system owned facilities.
2. A specification of all system owned facilities including services, tables, lists, etc., and the linkage or access requirements for each.
3. A specification of the logical flow between system modules.
4. System standards to be followed.

5. The file and directory structure for externally stored system information.
6. Overall size estimates for total system and core resident portion.
7. Execution time estimates for important paths, including I/O time.

For each module:

1. A brief summary functional description, size target, execution time estimate for principal paths.
2. Detailed specification of entry point linkage requirements, including all parameters required and the general format of these.
3. Detailed specification of exit linkages including destination, the parameters provided, and their general format. (Note: the latter can simply cross-reference a specified entry point in the destination).
4. Detailed specification of all system owned facilities used, including services, tables, list, etc., and the linkage thereto (as above the latter can simply cross-reference linkage requirements of these).

Machine processable specifications

A most useful tool in planning and controlling the changes to a large and complex system is a comprehensive where-used file for every module and communication facility in a system: that is to say, a file which for every such facility tells where to find every possible reference to it. Thus, when the designer plans a change to the system, he can tell immediately where the impact of the change will be felt. It is virtually impossible, however, to produce such a document unless plans are laid early in the design stage to generate the information required to make up this file.

This information should be contained in the specification for every module that is written. It should include every entry and exit point, as well as all the parameters that go to make up those linkages. It includes any references made to system facilities, blocks, tables, rolls, lists, etc.

These should be written on a formatted specification sheet which can then be put into machine processable form, and if desired, stored on disc or tape. A program can be written to scan this information to generate all cross-references and to test that these are consistent between the calling or referencing program and the facility which is called or referenced. Thus the system designers can have an early opportunity to spot cross-system communication faults in the specification.

This process can be carried a step further when actually coding such references if the programmer is required to use system controlled macros which, themselves, can be cross-referenced to the specifications. When the programs are coded and before they have actually been tested in the system, they can be scanned to compare macro cross-references with the specifications for the module to ensure that they are consistent. It is worth noting that these inter-module and cross-system errors are particularly difficult to debug; hence, the value of doing this sort of checking ahead of time should not be underestimated.

Implementation

Starting from a suitable specification, one should try to implement it by way of a systematic build and test process. This begins with programmers coding and testing modules with tests they write to satisfy themselves that the programs are working properly. These may be combined with other modules to form sub-functions which are in turn tested by tests written by the group responsible for the sub-function. However, because of the complexity of debugging a large system, it is essential at the early stages in system assembly to hand over these sub-functions to a central group, called *integration*; more about them later.

At this time these sub-functions will be tested by tests written, not by the programmers, but by a test production group. They are intended to verify that the sub-functions work at the level which is expected for incorporation into a system or sub-system which in turn is to be tested by system or sub-system tests written by this group. Once these sub-functions have been accepted they are combined with other functions and scaffolding, where appropriate, so that overall tests can be run.

Detected bugs are reported to the programmers who are given a copy of the system along with the test case or cases that failed, so that they can reproduce the problem.

The system against which overall tests are run also may be used as a master system by programmers working on functions that are not included (but perhaps scaffolded) in the master. When fixes have been found for the bugs in the master system at the current level (unfixed bugs may be converted to restrictions) and when the new function to be applied to the master system is available, then a new master system will be built using the old master, plus the fixes for its problems, plus the new function, see Figure 2. When this system has been built, it will be tested using a pre-specified newer and higher level of tests. The bugs thus found will be fixed or restricted, and it will become the new master system, available to programmers for the de-

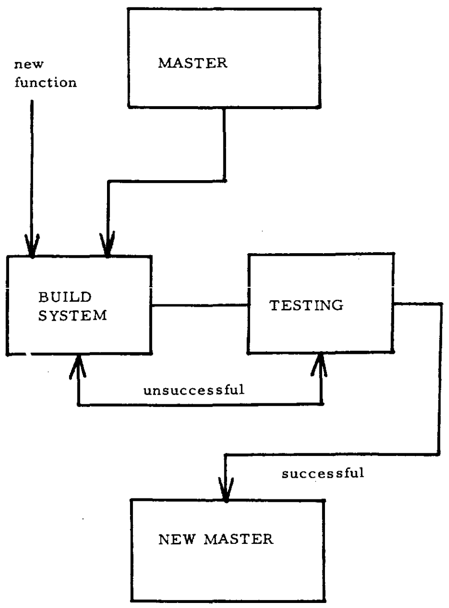


Figure 2—Build process

bugging of new functions. This process is repeated until all specified functions are incorporated and working in the system. Thus the systematic built process starts with a system at a given level, develops new function and fixes for the system at that level, applies these in order to create a new level of system which after test becomes the new master system.

Build plan

Given such an incremental build process one must be able to plan the points in time when given functions will be incorporated into the master system. For in most cases every new function has functional pre-requisites which must work properly. Thus, the build process requires a build plan which serves to synchronise all of the effort in the build process. This plan will specify the functional capability of the various system prototypes at every stage so that the programming groups will know what will be available and can thereby make their plans. Further, it will give the schedule on which these prototypes are to be achieved; it will specify when sets of test cases must be available to facilitate further debugging; and it will show a schedule of required hardware configurations to support the various levels of testing. Thus, the build plan describes the overall logistics and schedule of the build process. It is the central control document against which system build progress will be measured and in accordance with which all other system plans must be derived. As with drawing up the system specifications it will pay dividends for the system developer to take the care and time to lay out the build plan and then to make sure that everyone who is concerned with the build process understands it thoroughly.

Following the D-day landing at Normandy, General Eisenhower was asked whether all the planning that went into the invasion preparation was worth it, in view of the fact that practically nothing went according to the plans laid. His response is reported to have been that the plans themselves really did not prove very worthwhile, but that the planning process had been of infinite value. The same is true of building a large system. It is almost a certainty that there will be some unplanned chaos during the implementation period, but proper planning will prove invaluable in recovering from that chaos. It achieves the following:

1. Everybody associated with the process knows what are the critical factors and understands the sequence in which steps must take place. Thus, everyone is more able to assess immediately the effect of a change in plan on them and to re-align his plans quickly.
2. A widespread general understanding of the plan requires that all adopt common terminology and that meanings and implications of words will be thrashed out so as to facilitate much better communciations between everyone. Thus when changes in plan are discussed one can be more certain that everyone is talking about the same thing.
3. It permits those concerned to more readily identify the sensitive steps in a sequence so that contingency plans can be laid to back up the main thrust of planned operations.

Integration

Integration is the central organisation which carries out the build function in accordance with the build plans. They maintain programs in controlled libraries; from these they build systems in accordance with the build plan and cause these to be tested. They apply approved changes to the programs contained in the controlled libraries, and they provide systems at certified change levels to programming groups which require them in order to test new functions.

The integration libraries are the central store for the system; they service different groups of programmers who are testing and debugging various parts of the system in parallel. Throughout this period the attempt is made to maintain the rate of fixing bugs at the highest possible level, hence the change activities to these libraries is very high. Yet the changes applied to them must be stringently controlled to avoid regression, as discussed later.

Structure of libraries

One convenient way to structure integration libraries is to divide them into three sub-libraries, with different controls on each; one of them is the master library for the system at its current level; changes to it must be stringently controlled to avoid the application of changes which would either put the master copy at an unknown change level or would cause regression. The second library will contain a system or systems taken from a master library against which programmers may apply changes freely in order to determine proper fixes for bugs. The third, called the build library, is where integration is building the system at the next planned change level. That is to say, a copy of the current master library is being updated by carefully vetted and certified changes; when complete and tested it will become the new master system, and will be moved to the master library.

Regression

The scourge of this process is called regression, wherein: a fix is applied to a program, which may or may not fix the intended problem, but which produces problems or bugs elsewhere that did not exist before. Unless very careful control is kept over the changes applied and over the change level contained in a given library, regression is a virtual certainty. In a large system where change control procedures are not adequate it can hang like a malaise over the whole project. I have seen situations where on average for every bug fixed another one was created, so that over a period of time the net bug fixing progress was zero. The cure for regression is like the cure for rabies: it is painful, but it is guaranteed successful. Application of the following principles is required:

1. A single individual (in integration) must be designated as responsible for control of the libraries. He is responsible for setting up and operating all procedures for updating libraries and for authorising all changes to them. He should be made directly accountable for any regressions that occur. Practical problems such as working multiple shifts may require that he delegate responsibility for authorising changes to the library, but this should only be given to a few carefully chosen managers reporting directly to him.
2. Each bug to be fixed is described on a master list along with the manager responsible for providing the associated fix. The only fixes accepted as library updates are those which are on this list, when authorised by the appointed manager.
3. Any changes made to the build library should be the smallest modification which solves the problem which the change is intended to fix. Under no circumstances, short of absolute necessity, should changes be made to the library by replacement of whole modules or programs. Changes should instead be made on an add/delete basis; the libraries themselves, the programs contained in them and the programs which perform the updating must be designed so as to facilitate this. The integration manager authorising a change should have certification from the programming manager submitting it that it fixes a given identified problem and that it does nothing else.
4. Where possible, the integration manager authorising the change should review the code himself and be personally satisfied as to the extent of the modification. If he is in any doubt he must ask that the programming manager submitting the change review it with him before he agrees to accept it.
5. No other avenues for changes to the build library may be permitted.

The control procedures I have outlined are an attempt to prevent the application to the master libraries of unknown and often unwanted changes. These can arise through failure to keep an accurate accounting of the nature and scope of each applied change; through misunderstanding between the programmers and the integration people as to the functional level of the system in the library (and hence what changes are appropriate); and last, but not least, from purely altruistic motives of programmers who want to make improvements to the system that are not required to meet specification. Throughout the build process one must constantly strive to identify those problems which must be fixed, to fix them, and to avoid making other unnecessary changes.

The problem of controlling information in a system library is an order of magnitude worse than the problem of controlling accounts in a large bank. In the bank one has to keep track of how much money there is in each of a relatively few numbers of accounts and cash drawers. In controlling a system library one has to keep track of the value in a very large number of bit positions. This is rather like asking the U.S. Treasury to keep track of where every dollar bill is by serial number.

While the banks long ago adopted a very rigid accounting system to solve their relatively minor problem, I have seen installations where the whole control process was carried out on the back of an envelope. It is not normally essential for large system programmers to adopt the rigidity of banking procedures because the cost of error is not so high; I am suggesting, however, that the prudent application of a few strict controls can save time, toil, tears and money in the development of systems programs.

Testing

It is axiomatic that the quality of a program is no better than the quality of the tests that have been successfully run against it. Experience shows that those functions which have been tested and are known to work, will work; and those which have not been tested most frequently do not. It will be illustrative for the reader to recall how seldom he has taken what he thought to be a well established and fully exercised program and was successful the first time in running it in a new environment. The design of the tests and the laying out of the sequence in which they are to be applied to the various functional levels of the system is a most important part of the build process.

In designing a test procedure it is important to set aside the notion that the objective of testing is to achieve success. The real problem for the systems tester is to identify and isolate as many as possible of the most important bugs in the shortest time. It is most useful to regard debugging as a process in which one is trying by experiment to identify the bugs in a system. Just as in scientific experiments, the aim must be to derive the maximum amount of information for the least cost and time.

Design of tests

In testing a newly coded program, one would like to have tests that are function-specific; that is to say, they exercise only one funtion at a time in a way that does not call on other facilities and programs. On the other hand, the successful running of such a set of cases is not indicative of the true state of the program in its operating environment; since in normal operation of a program one function calls another. The most difficult bugs to find are usually those which result from interactions between modules or segments of programs when the individual functions seem to work but the combination of them does not perform as specified.

Therefore, the set of function-specific cases is not enough. Tests must be written which exercise cross-system capability as well. Ideally, one would like to have such a set of tests for every level of interaction in the system, but the economics of this are normally overwhelming. Thus a compromise is required to limit the size of the tests. Test planning and design is a most complex subject and one worthy of discussion separately. In this paper I only want to mention some straightforward but nevertheless important principles in systematic testing.

The test designer will work out the design of tests and the sequence in which they are to be applied with two criteria in mind:

1. In the event of failure of one or more tests then he wants to gain the maximum information from the test failure pattern
2. In the final stages of testing, it is necessary to apply tests which exercise the system in a comprehensive way. Prior to that, however, the test designer will want tests which are as independent of common functions as possible consistent with the need to test increasing functional interaction as testing progresses.

Thus, the first tests applied to a system each will separately exercise perhaps only one critical functional element of the system. These tests are used to check that these functions are actually in the system and working properly in a limited environment. When these are successful, then multi-function tests can be applied which aim to test both functional elements and their interaction in a restricted environment. Finally, full functional testing can be undertaken. When planning and designing these tests and their application sequence, however, the test designer should proceed in reverse order:

1. He should begin by deciding what will constitute the ultimate system test.
2. He will then work backwards to establish the multi-function tests required.
3. Finally, he can plan the functional element tests required to begin with.

Just as reproduceability is an essential characteristic of a valid scientific experiment, so an essential characteristic of a successful testing and debugging is the ability to reproduce reliably the occurrence of bugs in the programs under test. This can pose problems in interactive systems where the sequence of events depends upon the unpredictable sequence or occurrence of events in the outside world; thus careful thought is required to achieve reproduceability in these test situations.

Regarding the amount of effort one should put into test design and development, I would suggest that the proper level of effort in independent test development should be something between 15 percent and 20 percent of the system development effort, not counting early test writing by programmers to get their programs off the ground. Even in a system developed with a relatively small effort by a few people, it is important to have independently developed tests in order to ensure that the written specifications for the operation of the program .can be properly understood by someone besides the people directly involved in writing it.

System tests must be considered an integral part of the system; the experienced system developer in laying out plans for getting the system into operation will not distinguish between the need to get the system programs working and the need to get the tests working properly. The system will be no better than the tests applied to it; hence functional failure of the test should be treated in the same way as functional failure in the system.

The test cases must be held in libraries under the same controls as the system itself, so that once tests are running properly, their integrity can be guaranteed.

CONCLUSION

In the foregoing I have outlined some of the basic principles that I feel are important in the designing amd implementing of large systems. The principles I have discussed are only the beginning; I believe there is broad scope for innovation in management and control techniques which aid large system development. But I believe that use of these techniques can free designers and programmers from the severe strain of control by pure discipline. This will not only make the development process more predictable, but will make it faster, cheaper and more rewarding for those involved.

REFERENCE

1 P H HARTMAN D H OWENS
How to write software specifications
Proc F J C C 1967
Hartman and Owens in this paper discuss many facets of compiler specifications:
* User orientation
* Performance, size
* Optimization
* Debugging
* Statement of intent
* Acceptance tests
* Specification maintenance

Management of computer programmers

by MALCOLM H. GOTTERER*

The Pennsylvania State University
University Park, Pennsylvania

INTRODUCTION

The problems of managing programmers have been growing rapidly in recent years. The specific issues referred to are the increasing costs of programming and the failure to complete projects when scheduled or needed. This paper proposes more effective supervision of programmers together with improved management policies and procedures as steps to reduce the effects of these growing trends in programming projects. These problems have resulted, in part, from the shortage of qualified personnel, the rapidly changing technology, and the management personnel in the computer center. This paper does not deal with the problem of project management, that has been dealt with elsewhere,[1,2] but rather the supervisory processes involved in the management of computer programmers.

The shortage of qualified personnel

One of the fundamental problems facing the manager of a programming group is the shortage of qualified personnel. The number of installed computers is rising rapidly with a variety of estimates being made for significant future growth. Regardless of whose estimate is accepted the fact remains that an increasing number of computers will be installed and operating in the near future. We cannot expect, in the short run, to find a decreasing demand for programmers. It has been estimated that at the present time there is a shortage of over 50,000 programmers in the United States.[3] A recent major study of the problems of the computer industry estimated that by 1980 there will be a need for 200,000 programmers.[4] These figures have led Brandon[5] to estimate that by 1980 there will be a net shortage of about 90,000 programmers. Part of the staff shortage problem would be relieved if the educational institutions were to provide the proper education to a large number of students. In a study of computers in higher education Hamblen[6] found that in 1964-65 nearly 120,000 undergraduates and 29,000 graduate students received some computer training. In that same time period it was estimated that approximately 4,000 undergraduates and 1,300 graduate students were majoring in computer science. By 1968-69 it was estimated that 81,000 graduate students and 350,000 undergraduates will have training in at least one programming language. At that same time the number of students whose major area of study is computer science will increase to 19,000 undergraduates and over 5,000 graduates, see Table I.

Table I—Estimate of the number of students being trained to use computers

MAJOR	COLLEGE LEVEL			
	Graduate		Undergraduate	
	1964–5	1968–9	1964–5	1968–9
Computer Science majors	1,314	5,318	4,338	18,807
Other majors (with at least some skill in using one programming language)	28,800	80,793	119,092	350,178

Source: Reference 6

The response of the educational institutions to a national problem, as reflected in these figures, is impressive. But before concluding that in a few years the problems of shortages will be resolved by adequate supply it must be realized that there will be intensive

* Currently on leave of absence at The Johns Hopkins University, Baltimore, Maryland.

competition, as there now is, for those students who have majored in computer science. In addition it will require substantial incentives to motivate the over 400,000 students who have had training in at least one programming language to leave their own field of major interest and become programmers. Even for those who are lured into computing further intensive training will be needed to bring them up to an acceptable level of competence. Therefore it may be found that in the next five years a labor pool exists of college graduates with some training in programming. The shortage of qualified and experienced programmers will still exist and perhaps be even more severe due to the need for training and supervising these new entries into programming. There also has been a rapid expansion of non-degree granting programming schools both private and public. It is too early to appraise the effect this source of supply will have on the labor market.

It should be noted however that even though we can expect staff shortages to exist in one form or another for some time to come there is something that can be done to relieve the problem. That is to increase the productivity of programmers. If programmer productivity were increased by a modest amount, say 10 percent, it would, based on estimates of 200,000 programmers, be the equivalent of 20,000 additional programmers being made available. This would be 40 percent of the estimated current shortage of 50,000 programmers. The techniques suggested in this paper have been used to increase programmer productivity and it is toward that end that they are presented.

Rapidly changing technology

The past changes in computer technology are well known and need not be repeated here. Of importance, however, are the changes that may take place which effect the technology of programming. Such technological developments include languages, compilers, operating systems, documentation aids, etc. In the past a change in hardware technology has generally brought with it changes in the programming technology. The result is that additional time and money must be invested for training so that the staff can stay abreast of the latest developments. These types of changes are mandatory because the new hardware technology has forced them on the user of the particular computer. A second category of optional, or non-hardware related, changes exists. These are those technological developments which result from attempts to achieve maximum exploitation of the computer. The training of programmers in these techniques is not mandatory but frequently is highly desirable. The net result of these two types of changes in technology is that continuous training for programmers is a way of life for effective program production.

Even now there is some indication that the next generation of computers may not be program compatible with those currently available and in use. An executive of a major computer manufacturer has been quoted in an article as saying that his company would not hesitate to ask the data processing community to "reset-to-zero" and accept program incompatibility "if something comes along we think is a substantially better way to get price/performance into the market."[7] It therefore appears that still another cycle of programmer retraining and program conversion may be forthcoming.

The computer center manager

Fundamental to the solution of the problem of better management of programming efforts is the manager of the computer activity. By computer manager we mean the highest level person in the organization who devotes all of his time and energy to the problems of computation. This definition avoids having to enumerate the myriad titles associated with the position. In a study of the managers[8] of large computer centers it was found that these managers were relatively young (41 years old), well paid (average salary of over $20,000 per year) and well educated (typically a second degree). Of significance is the fact that only 12 percent of those sampled preferred only technical problems, 40 percent preferred a combination of technical and human problems, and 48 percent indicated that the principal problem they preferred was human. Educational background had a slight effect on these average results, Table II. When asked which problem they face gave them the greatest difficulty 42 percent indicated personnel, Table III.

One of the reasons that can be attributed to the manager's personnel problems is the rapid growth of the industry. As a result of this growth many persons have been forced into managerial positions without adequate preparation. This lack of prior managerial experience is most noticed when it is observed in a number of computer centers that there are inadequate control systems for reporting on the operations of the department or organization. The quintessence of management control is the planning of activities, delegation of authority, staffing the organization and control of plans over a period of time. This has not been generally done in computer organizations.

It is therefore proposed that the problem of managing computer programmers starts with the senior manage-

Table II—Education vs. problem preference

	TYPE OF PROBLEM PREFERRED			
Education	Technical	Human	Both	Percent in Group
Technical degree or degrees	14%	50%	36%	56%
Technical degree plus M.B.A.	0%	100%	0%	4%
Business and Liberal Arts Degrees	20%	40%	40%	40%
Percent	12%	48%	40%	100%

Source: Reference 8

Table III—Problems facing the computer manager

PERCENT	*PROBLEM*
2.6%	Lack of Funds
42.2	Personnel
18.4	Education of Top Management
18.4	Keeping up with User Needs
15.8	Long Range Forecasting of Computational Requirements
2.6	Cost Control
100.0%	

Source: Reference 8

ment of the computer center. These managers are faced with the usual dual technical and human problems. But the technology is changing so rapidly that it is difficult to fault them for indicating an interest and concern about personnel and then actually observing them more concerned with the effective operations of systems and machines where semi-technical considerations (such as operating systems) occupy an inordinately large amount of their time. Further we find in many organizations that the manager has not had the time, resources or both to develop effective secondary levels of management beneath him. In this case no matter how effective he might be the number of activities he must control, because of a lack of subordinate managers, or inadequate subordinate managers, is so high that he cannot devote the time necessary to the tasks of planning, controlling, and counselling. One of the first steps in improving the management of programmers is therefore to improve the calibre of programming managers.

The programming manager

Unfortunately little is known about programming managers. A great deal can be conjectured based upon an examination of the individuals performing this job. First, and perhaps most important, is the process by which a person may be appointed to this position. The logical, but unproven, concept that the best manager is the one who has done the job for a reasonably long period of time leads many to look to their most experienced programmers to find a new manager, or supervisor, of programmers. Little or no attention is paid to the individual's suitability for a position of leadership. The individual, on the other hand, is faced with the dilemma of leaving technical work for the higher pay and status of a management position. Few firms offer the much talked about dual career paths for professionals and managers. Even among those who do there are frequently differences in salary ceilings for the person who chooses technical rather than managerial work. The result is that the first rate programmers are frequently motivated by financial factors into becoming second rate managers.

Steps toward change

While it is possible to identify a number of causes of present, and probably future, problems there is no magic formula for an effective solution. It is proposed that through the intelligent and orderly application of management techniques steps can be taken to reduce the impact of these problems and provide a basis for further improvements. The point of view taken is that the manager of a group of programmers is responsible for the *quantity* and *quality* of work produced. He is also responsible for *meeting schedules.* The techniques available for him to use in achieving these goals center about the management process in general and include such topics as supervisory techniques, work assignments, evaluation of out-put, documentation, work habits, leadership, and evaluation and review in particular.

The management process

Reynolds[9] has pointed out that while the management of programming is similar to the management of other technical activities it differs in that there are few numerical indices that can be used to judge progress in a programming project. There are the following steps in the management process:

1. Establish a plan of action to achieve a goal within a certain time period, and investment.

2. Measuring, during the course of the project, the performance against that goal.
3. Making evaluation and decisions each time the plan and the actual performance mismatch.
4. Taking corrective action iteratively until the job is done.

The establishment of a plan requires that there be sufficient understanding of the task so that it can be broken down into sub-tasks and using a procedure such as PERT an overall strategy developed. If this level of knowledge about the project is not known it must then be considered to be a research and development effort and so treated.

The second step, that of measuring during the course of the project actual performance and comparing it with the planned achievement, depends on an effective reporting procedure. The system of reports can be related to, or even a part of, the system documentation as is frequently recommended.[1,2] Reports are meaningful only if they are related to the benchmarks or steps used in the planning process. For evaluating the effectiveness of the ongoing programming effort the use of PERT/COST has proved of value.

The last two steps of the planning process follow logically after the data is available. What steps are taken should performance not equal that anticipated by the plan is dependent on many factors. There are, of course, good reasons for deviation such as changes in specifications but there are also reasons such as failure to produce at the expected level which represent problems in personnel management or estimating. The corrective steps taken are a function of the problem and its likely cause. In these last two areas the manager's knowledge and ability come to the foreground. Since the first two steps are somewhat mechanical and have been dealt with elsewhere attention can best be focused on the problems of the manager in his supervision of programmers.

Supervision

The point has been made that the management of programming is another example of the more general problem of the management of technical personnel. But when this general statement is made it is often forgotten that the management of a technical function usually requires two skills. The first is an intimate knowledge of the specialized skills involved and the second is a knowledge of more general managerial skills. In such an environment as that under discussion it is questionable whether an effective job can be done with only the second. Too many have tried to be effective with only the first, technical skills, only to find themselves faced with many difficulties. The programming manager, like any other manager, requires the combination of these two types of skills to perform his job.

Programming is still largely an art rather than a science. The programmer therefore learns his trade from experience either on the job or in a classroom. In either case there is only a small, but hopefully growing, body of theoretical knowledge that is available. The programming manager must therefore have served an apprenticeship to have learned his trade. These specialized technical skills, it will be seen later, are essential for the manager of programming. The general management skills required can be learned in the traditional fashion by attending courses, on the job training, tutorial sessions with experienced managers, and others. While it must be acknowledged that few men become managers as a result of courses and training alone it is also true that there is a body of knowledge about management that can efficiently and effectively be taught. A well instructed individual can then apply this knowledge and start on a management career.

It has been emphasized in this paper that the programming supervisor is responsible for programmer management and productivity. We therefore now examine some of the steps he can take to insure that these goals be achieved.

Work Assignments. The programming manager should define the job to be done by each person under his control. Ideally this work assignment will be a well defined program step which is to be written, tested, and documented. A good work assignment will require minimum interaction between programmers implying that interfaces, fields, data names, and other such matters have been defined prior to any work assignment having been made. Further, the programmer should be expected to confer with his supervisor on any question of technical details that arises.

It is not always easy to divide a large and complex program into modules that can then be assigned to a programmer. Alt[10] contends that it is the inadequacy of natural language that creates problems for the programmer because it is not a good language for communicating and specifying all possible combinations of circumstances that relate to the problem. The formalisms for problem definition and description which are required are not now available. As a result we must be aware of the problem and continue to use the module as the work assignment.

Evaluation of Output. The manager has the responsibility to evaluate both the quantity and quality of work produced by his subordinates. To do this the supervisor should review the programs that are written. In this way he can be aware of the amount of code written and he can evaluate the quality of the work. In these evaluation sessions the manager can tutor the programmer, as required, in more advanced techniques or alternative approaches. The implication of this is that each programming supervisor have relatively few subordinates whom he can closely supervise.

It has been proposed that productivity standards be used for purposes of evaluation.[11] From a practical point of view this is difficult because all the variables that affect productivity are not known. Martin[12] has pointed out that while there has been found an average figure of 12–15 instructions per man per day the range is from 2 to 50. Even if standards were available Canning[13] questions their use from an emotional point of view. He points out that they would place the programmer in the same category with factory and clerical workers. Undoubtedly this is true but the needs to increase productivity and reduce costs are as great in programming as in the office or factory. When such standards are developed they will be one of the important tools of the programming supervisor. At this time it is difficult to envision how a meaningful and accurate set of such data could be collected.

Documentation. Good documentation of a computer program is not an activity to be tagged on to the end of a project. Rather it is an ongoing activity with documents being developed at each stage of the programming process. The manager, therefore, is responsible for establishing documentation procedures and then perhaps more importantly enforcing them. It is interesting that Hill[14] reports that resistance to documentation is encountered by all but the most experienced programmers. Those with one to four years experience being the most resistant. If the programming department has standardized its documentation procedures then the supervisor's principal function in this area is to make certain that it is being properly carried out.

Work Habits. Amongst some programmers it is considered good practice to flaunt any or all work rules. The argument is advanced that programming is a unique creative activity that should not destroy personal values. The actual amount of creativity involved in programming is questionable. Further there is no reason why work rules should destroy initiative and creative urges in programming any more than in other comparable fields of activity. In many architectural and engineering offices reasonable work rules have been established and enforced and some of these people turn out very worthwhile work. The programming manager can be expected to establish work rules which are then enforced.

Leadership. Perhaps the most difficult job of the supervisor is to lead his subordinates. Leadership processes can be discussed in classes and written about in books but it is difficult to define. Each supervisor must develop a leadership style that fits his personality and life style. He can be made aware of the need for leadership and be told about how others have done it, but in the final analysis this is one of those intangible factors that each person must develop on his own. Failure to do so should result in a serious question as to whether or not the person is qualified for a higher position or even to continue in his present job.

Employee Evaluation and Review. It is common practice in many organizations for supervisors to meet with each subordinate and review the persons performance for a period of time. The review may take place annually or semi-annually but it is required and is then reviewed by managers higher in the organization. Formal employee evaluation is relatively rare in programming departments and should be encouraged. To aid the evaluator a form is used which can serve as a checklist to guide the discussion and provide a permanent record of the interview. One such form has been developed and published.[15]

Employee reaction

It has been said that employees welcome controls such as those advocated in this paper. With programmers this is not true, resistance, perhaps even massive resistance, can be expected. Part of this resistance is based on the folklore that has grown up about the field of programming and the behavior patterns that are legitimate for programmers. At one time in the history of computation it might have been true that such employee controls were unreasonable. But today such supervisory techniques when used with intelligence and understanding are not only desirable but essential for controlling activities.

CONCLUSIONS

The program of management controls recommended in this paper have as an objective to increase programmer productivity. Collectively they imply firmer controls over the actions and behavior of programmers. Undoubtedly some programmers will be unwilling to accept this condition and will leave. Their effectiveness will be largely a function of the intelligence with which they are used. In all cases it is recommended that all processes discussed be applied. But the emphasis may well change with the experience of the people involved and the type of programmers being supervised. Finally these recommended procedures are complementary to, and not substitutes for, good project control procedures.

REFERENCES

1 C P LECHT
The management of programming projects
American Management Association Inc New York 1967

2 P B HILL
The control of large-scale software projects
Paper presented at IFIP WG3.2 Workshop on Curriculum For Systems Design Fribourg Switzerland January 28–31 1969

3 G BYLINSKY
Help wanted: 50,000 programmers
Fortune Magazine March 1967

4 *The state of the information industry*
AFIPS Report 1966

5 D H BRANDON
Managing the economics of computer programming, the problem in perspective
Proc A C M Conference 1968

6 J W HAMBLEN
Computers in higher education
Southern Regional Education Board Atlanta 1967

7 *4th generation may be incompatible*
Computerworld February 12 1969

8 M H GOTTERER
A profile of the computer manager
Proc Fifth Annual Computer Personnel Researh Conference 1967

9 C H REYNOLDS
Managing the economics of computer programming, the problem in perspective
Proc A C M National Conference 1968

10 F L ALT
Computers—Past and future: The costs of computing; and failure in computer programs
Computers and Automation January 1969

11 D H BRANDON
Managing the economics of computer programming, the problem in perspective
Proc A C M National Conference .1968

12 J MARTIN
Programming real-time computer systems
Prentice-Hall Inc 1965

13 *Managing the programming effort*
EDP Analyzer Canning Publications June 1968

14 P B HILL
The control of large-scale software projects
Paper presented at IFIP WG3.2 Workshop on Curriculum For Systems Design Fribourg Switzerland January 28–31 1969

15 R A DICKMANN
A programmer appraisal instrument
Proc 2nd Annual Conference Computer Personnel Research Group 1964

The management and organization of large scale software development projects*

by RONALD H. KAY

IBM Research Laboratory
San Jose, California

INTRODUCTION

Two consecutive papers on the subject, "Managing the Economics of Computer Programming" presented at the 1968 National Conference of the Association of Computing Machinery conclude respectively:

- "First, one must understand computer programming well enough to know what is possible, what is probable, and what is impossible or unlikely.
- Second, one must make commitments based on the technology used, not on the needs of the world—and not on the unreasonable hopes of the starry-eyed experts.
- Third, one must insist upon schedules based on physical events, and on numerical descriptions of the products that are being produced, to the greatest extent that ingenuity will permit.
- Fourth, one must objectively assess the status of the project against a well-developed plan.
- Finally, of course, one must do something about the trouble one finds.

Thus, given these prerequisites, I conclude that computer programming can in many respects be managed just like any other process."[1]

"We do not really know how to select programmers, and we tend to select those with some undesirable characteristics. . . . Typically, they work for a manager who is ineffective because he has been given neither proper management training nor basic tools and disciplines with which to work; whose functions have not been defined, and whose process of communication with the system analyst or user is generally confused. Finally, all this takes place within a technology which changes so rapidly that it is almost impossible to get a fix on the functions and the method by which the work is supposed to take place, before it changes."[2]

An equally wide range of views emerged from a series of seminars organized by Professor Maison Hare of the Sloan School of Management and Professor Malcom Jones, Assistant Director of Project MAC during the fall of 1968 at MIT.

Insofar as the seminar speakers represent independent organizations with widely varying objectives, divergent views on many issues are hardly surprising. Management objectives will be viewed differently by a software firm working on a contract, a computer manufacturer developing an operating system for his hardware, or a university attempting to develop new concepts in time sharing systems.

Thus, the basic contention that the management of large programming efforts does or does not present a unique problem, suggested in the above quotations, may reflect the relevant management experience which an organization has been able to bring to bear upon a problem, rather than conflicting conclusions drawn from a given set of premises. Each of the invited speakers* had extensive experience in the management of large programming efforts and is, or had been, associated with an organization presently involved in such efforts. Most of the organizations represented have won recognition both, for leadership in technical innovation as well as success in the large scale applica-

* Based on a seminar series attended by the author while a Fellow at the MIT Center for Advanced Engineering Studies, Fall, 1968.

* The names and affiliations of the participating speakers are appended.

tion of new technology. Unfortunately, some of the organizations active in the advancement of this field of management and whose work was referred to by several speakers, were not represented.[3]

It must also be pointed out that most of the speakers were executives with direct responsibility for large programming efforts or for all programming activity in their organization rather than specialists in management science concerned with the advancement of this field.

It would be presumptuous to attempt a condensation of these seminars representing the experience and wisdom of the 21 speakers and the reactions of the other participants. Rather, this paper endeavors to find common threads and possibly resolve conflicting views on specific issues. It is based entirely upon the content of the seminars, and represents the author's interpretation for which he assumes full responsibility.

This interpretation reflects the relative weight given by the speakers to various aspects of the subject, with some additional emphasis where there was significant disagreement among the contributors.

Since the more visible sources of management difficulties are often the derivatives of unclear or changing objectives, this issue is identified at the outset. Next, the need for accurate assessment of the state of the art, the difficulties of making such an assessment and its relevance to the project organization are considered. Arguments are advanced for two kinds of organizations based upon the objectives and the required degree of innovation.

The scope of this paper permits only the most general observations regarding techniques for project planning. The significance of the project plan as the principal means for rendering a programming effort visible is pointed out. The discussion of specifying and evaluating a complex programming system and measuring the performance of the people responsible for its development serves to illustrate the magnitude of the problem faced by management at this point in time. The lack of a common set of tools and standards are pointed out as some of the specific causes of difficulty.

The issue of higher level languages is still an attractive topic for debate; those in favor appear to be pulling ahead. This analysis of presently held views concludes with the observation that recent advances in the shared use of computers through interactive terminals promises to provide programming development management with more effective means for the control of the implementation phase by putting documentation online.

The basic assumption underlying the seminar series, i.e., the existence of significant unresolved problems was confirmed. There is little contention that frequent underestimation of the risk, possibly more than in other areas of innovative development, is due to the disproportionate responsibilities placed upon programming management relative to their experience and the maturity of the field.

The term "programming" is here used in its broadest meaning to include all phases of software development. In emphasizing "Large Scale Software Development Projects" the seminars focussed principally on complex operating systems, command and control systems and large simulation efforts, in each case involving groups large enough to require several levels of management. The systems referred to typically involved 100,000 to several million instructions.

Objectives

The statement of definitive and time-invariant functional objectives at the outset of a large programming effort has proven to be one of the requirements most difficult to satisfy. For example, the compatibility of an operating system relative to a family of hardware or the extendability of a storage/management approach to a multi-processor system are objectives difficult to formulate at the outset of an effort intended to break new ground in these areas. In spite of this, management has frequently accepted vague or unrealistic objectives in the expectation that the project itself will produce the required clarification or advances.

Inadequate objectives at the outset, more than any other single factor, are held responsible for subsequent modification and consequent overrun of initial schedules and budgets. It is felt that this situation will continue as long as management or the sponsoring agency feel the risk to be commensurate with the potential benefit.

Why are objectives held to be more vague and subject to change in the case of large programming efforts than in the case of hardware development? Certainly, the relative experience accumulated in the two areas favors definition of hardware. Even when objectives are dictated by non-technical considerations, those charged with the responsibility for setting these objectives are more familiar with hardware development. They may avail themselves of advice from programming experts but their judgment of possible alternatives is still influenced by their experience.

From the views expressed by several of the seminar speakers, one is led to conclude that as long as the economic motivation associated with hardware is greater or thought to be greater than that associated with software, the latter will be adapted and modified.

Although only a few of the speakers explicitly referred to the influence of the organization's long term objectives such as profitability, personnel policy and the organization's image, it is evident that they can have a significant bearing upon the management approach. The shortage of experienced programming development managers who can effectively implement such objectives and in some cases, the apparent reluctance to face or discuss the issue of objectives tends to intensify potential difficulties.

The matter of objectives has been emphasized since the more visible sources of management difficulties are often the derivatives of unclear or changing objectives.

Assessment of the possible

While objectives are elusive, the outcome is predetermined by the state of the art. A particularly useful definition of the state of the art differentiates between three levels: (1) what is possible for the experts; (2) what is generally known; and, (3) what has been done by a development organization. This definition implies that expert knowledge must be relied upon to assess the degree to which the objectives depend upon contributions by expert personnel.

The first thing insisted upon by the experts is that there is no substitute for a basic conceptual framework which stands the test of time. That is to say, if the basic concepts and philosophy of implementation do not stand up, no amount of administrative technique can be successfully brought to the rescue. The lack of scientific discipline in the field, i.e., the absence of first principles upon which to base first principle calculation, so successfully applied in the physical sciences and engineering, produces the dilemma of conflicting expert opinions. The field has as yet not produced an adequate number of people who can match their managerial skills with this sort of expertise (or vice versa) to be able to resolve such conflicts with confidence.

Experts can be relied upon more readily for an assessment of what is generally known, or more precisely, of the advances required in what is generally known in order to meet the objectives. Frequently, the need for this implied precision is inadequately appreciated in the assessment by expert and manager alike.

It is only natural that the current literature on the management of programming efforts is more fruitful in the realm of what has been done by a development organization. There is persuasive evidence that many organizations have the competence to manage even complex tasks which require only minor advances in the state of the art.

Organization

The assessment of the possible in terms of three levels of the state of the art suggests that the organization of the project should reflect the objectives as well as the required degree of innovation. Management problems and compromised objectives have been traced repeatedly to situations where the resulting programming system reflects the a priori organization chosen for its development.[4]

Accepting the need for tailoring the organization to the objectives, the case has been made for two very different approaches.

1. A relatively small group of experts and selected support personnel charged with carrying the project from inception to completion.
2. A small group of experts in an advisory and monitoring function to the management of a large development organization where distinct groups of people are responsible for various phases of the effort such as analysis, design, implementation, integration, testing and maintenance.

Although these two approaches are frequently argued on their absolute merits, more often than not they are born of necessity.

A one-time commitment on the part of a university to a large programming system effort is often justified on the basis of an available small group of experts and their ability to muster a temporary support group with better than average qualifications.

An industrial organization committed to continuing development activities requiring varying numbers of people for a variety of tasks on a continuing basis, finds it necessary and effective to develop specialized centers of competence so that a number of projects can draw upon this resource.

It appears that the larger the relative need for innovation at the expert level, the stronger the preference for the small group. Given these two types of organization, the management techniques which find favor differ greatly. The large contract software organization may respond to the first sign of a problem by getting machinery in motion to hire or transfer an additional 100 programmers to the project. The rationale for this approach is that it may take six months to really understand the nature of the problem. At that time you probably have between 10 and 100 people with up to six months hands-on experience from whom you can select a few who can now be identified as being able to correct the problem.

The small group of experts, developing a complex system would react differently. Having less of a communication problem, they would identify the cause and come up with a potential solution in a shorter period of time. But by definition they would expect to make use of the same small number of people to correct the situation accepting the unavoidable postponement of other planned activity. Depending on the magnitude of the fix required, this could be an appreciable fraction of the original estimate. It can also be concluded that the small group is less able to absorb some of the influences beyond its control, such as, turnover of personnel, limited machine access, etc.

Although some of the speakers attempted to address the organizational issues in reference to the classical distinction between project and functional organization, it became evident that the definition of the various functions of software development in the represented organizations are not sufficiently precise or uniform to allow for meaningful generalization.

Depending upon the priorities of management objectives, such as minimum cost, fixed deadline or optimum performance, there is some basis for choosing an appropriate organization: ability to muster resources, degree of innovation required and long term objectives such as developing skills vs. hiring experienced people.

The wide divergence of views on this subject reflected the difference in management objectives and philosophy of the organizations represented.

The project plan

A significant amount of attention was given by a number of seminar speakers to the subject of a project plan. Such a plan identifies various phases of the project such as Analysis, Specification, Design, Implementation, Integration, Testing, Publication and Maintenance.

We shall here confine ourselves to some general observations regarding such a plan.

- The plan is not an end in itself, but a management tool which helps define responsibilities and checkpoints. It is the principal means for achieving visibility of the project.
- There is considerable overlapping of the various phases of the plan in the case of programming development; e.g., testing is initiated with the specification phase, where component-, integration-and acceptance-tests must be defined.
- Such plans point up a basic difference between hardware and software development. Hardware development, culminating in a tested prototype, leads to the subsequent manufacturing phase which generally requires much greater resources and thus becomes a major factor in the ultimate success of the project.
- Programming development culminates in the end-product and in this sense resembles the development of one-of-a-kind hardware.
- The use of PERT charts is generally held to be ineffective as a means of planning and control of large programming development tasks.
- Project control against a plan is not unique to programming projects.

Problems frequently are not due to a poor plan but to the fact that the plan is not being carried out for a variety of reasons, some of which are being considered in this paper. Above all, a good plan is no substitute for poorly defined objectives.

Specifications

One of the most difficult aspects of programming development is the process by which the results of analysis are translated into a set of specifications. What is wanted is a set of blueprints which uniquely specify what is to be implemented. The first problem arises with the decision as to what should go onto which blueprint. The need to break the job down into separate modules forces early decisions regarding the interaction between the modules. An attempt to expalin the concept of "Functional Modularity" may help to clarify this issue. In the case of hardware, modularity is derived from considerations such as physical dimension (what one can get through a door, tolerable delay, etc.) components which can be shared (e.g., power supplies) ease of access and replacement, standardization of modules, etc. Once a hardware module is defined, its relation to other modules is fixed by virtue of a finite number of interconnections. Every physically accessible connection is a potential test point permitting isolation of modules. In a complex programming system modularity is sought in terms of frequently used sections of the program and elements which are common to several functions. There is a desire to minimize interaction between modules and to achieve clean separation of function to facilitate division of the development effort and module testing. This concept of modularity as yet does not take account of the fact that not all parts of a complex program can be equally accessible at all times, i.e., sections of the program must be moved to provide the desired access. It is fairly obvious that the larger the "module" which is moved the fewer the required moves. Yet, a large module occupies more prime space and takes longer to move.

This suggests that the various considerations which influence modularity in a large programming system are functionally interrelated in a much more complex way than the parameters which influence hardware modularity. Thus, to achieve "Functional Modularity" at the specification stage, implies the need for past experience which proves relevant to the problem at hand or a methodical approach to reduce the inherent complexity. Neither appears to exist in the case of large programming development tasks, particularly where new ground is to be broken. This most abstract aspect of programming development remains the intellectually most challenging.

Another problem relating to programming specifications involves the degree of detail necessary. It is argued that to assure equivalent results from two programmers given the same specification, a level of detail (and effort) nearly equal to the program itself is required.

Reliance upon experience which may or may not prove relevant and the lack of a methodical approach hardly ease the problem of evaluating the end-product against a set of specifications.

Evaluation of the end-product

The end product of the development is a programming system, i.e., a collection of programs designed to perform a specified function in conjunction with specified hardware.

For example, a large systems program must, among other things, provide the scheduling and allocation of system resources as called for by a particular set of instructions, i.e., by the application program. What constitutes a "typical" application program or set of programs which would provide a realistic measure of performance?

Another aspect of evaluation is the data dependence of the program. To illustrate this point, consider the logical combination of two data elements whose combined value exceeds the capacity of some hardware facility. The programming system must be designed to cope with this problem in terms of all possible logical combinations of all possible data elements with regard to all possible combinations of affected hardware components. How can this be done? Only the most careful design of test programs and the most extensive test cases can hope to provide a satisfactory approximation to "all possible combinations."

Given this level of evaluation, what can we say about the quality of the programming system in its ability to cope with a given job stream which has a unique sequence of programs? How representative is this given job stream of the types of applications a variety of users are likely to encounter? How do the answers to these questions relate to a multi-processing or multi-user environment? What is more, how can we evaluate such objectives as compatibility and useful generality which are related to past and future hardware and applications?

To date, there simply are no generally satisfactory answers to these questions. Even when satisfactory answers are obtainable for a specific subset of the desired range of parameters, the evaluations upon which these answers are based can only be attempted *after* successful integration, i.e., when the job is presumably done.

While techniques have been developed which provide a basis for predicting the mean time between failures of hardware components, the asymptotic nature of program debugging and the effect of transient causes of error upon program behavior introduce elements of uncertainty which are difficult to quantify.

Evaluation of a programming system, probably better than any other aspect of the development cycle, illustrates the level of complexity and the relative lack of proven techniques in this field.

Evaluation of the programmer

The difference in approach to organization and the difficulty of evaluating the end-product highlights one of the unique problems of programming management: The measurement or evaluation of a programmer's effort.* There is conclusive evidence that there are order of magnitude differences in individual performance by almost every criteria, such as time required to complete an assignment, tightness of code, quality of documentation, running time, storage requirements, and computer time required for debugging. These criteria serve as indicators, but realistic measures of performance based upon these indicators are qualitative, not quantitative. These indicators would not necessarily point up interfacing problems before integration, or provide a measure of ultimate performance but would serve only as a warning in the case of extreme departure from the norm.

Two principal reasons for the large variation in performance among programmers and the difficulties of measuring this performance are considered in some detail:

- The craft-like nature of programming.
- The personality traits of programmers.

* "Programmer" is here defined to include all personnel engaged in the analysis, design, implementation and testing of computer programs.

Programming—Its craft-like nature

Non-programmers find it difficult to understand how a task, at once requiring the utmost in logical consistency, at the same time can provide so much choice in the approach to a given problem.

Hardware engineering experienced the same problems in its evolution from the skills of the craftsman to the mature technologies based on various branches of science. Today, two hardware engineers given a task to perform generally can agree on what constitutes a precise definition of the problem and what constitutes an adequately tested and documented solution. Even though their end products may look different, there will be considerable resemblance in their approach. They will go through clearly-identifiable prodecures such as analysis based on a set of equations, circuit diagrams, breadboard models, tested models, prototypes, etc.

In programming, on the other hand, the situation is different. There is generally no way to relate the work of two programmers or even the same programmer's work on two different jobs. Unlike engineering, the road from gross program design to a detailed design is a function of a set of highly unpredictable human events. As the job progresses, its nature tends to be redefined as the programmer becomes more familiar with the problem. The way he reacts to this increasing awareness and translates his reaction to the program is highly idiosyncratic to the individual and to the individual project.

Programmers are different

In listing the personality traits of programmers as a source of difficulty in the measurement of their performance, it must be understood that the difficulties referred to are those perceived by management. Many programmers probably accept the fact that they are not easily measured.

What is the basis for the assertion that programmers are different?

First of all, most managers who are led to this conclusion compare programmers to engineers.

Second, many programmers are recruited from the ranks of liberal arts graduates, while the managers were trained in engineering or business administration.

Third, in many organizations the programmers are the most homogeneous age group. They are thought to represent a readily-identifiable group of young people, bringing the new look from high school or college into certain segments of industry and government. This new look is sometimes equated with appearance and attitudes designed to set themselves apart from the existing majority.

Finally, many aspects of programming require a high degree of concentration over an extended period of time which tends to make programming a solitary occupation; those drawn and devoted to it may be or become more introverted than the majority of their nonprogramming colleagues.

Although management training in the liberal arts has long been advocated, few present day managers are prepared to cope with the generation gap or the culture gap. Until enough managers can be drawn from the ranks of programmers, the problem is likely to persist.

The tools of the trade

Most craft-like processes which have been carried on for some time evolve a set of tools which are available to the community of craftsmen. Their skill level may vary, but their tool kit is generally the same. In the case of large programming systems, and particularly when major advances in the state of the art are to be incorporated into the system, adequate tools may not exist. In fact, the evaluation of the adequacy of available tools or the creation of such tools can constitute a significant aspect of the project. The adequacy of the computer and the operating system available to the development effort is a case in point. Higher level languages and the adequacy of the available implementation (compilers, etc.) fall into this category also.

Another unique problem of programming development is pointed up by the lack of a programmer's equivalent to an oscilloscope, this most useful of electrical engineering tools. The core dump which reflects the contents of storage locations at a given program step is the nearest equivalent. It roughly corresponds to a simultaneous presentation of the wave form of every possible test point in a circuit ordered according to the geometric location of solder joints rather than points on a diagram.

It takes a great deal of experience for a programmer to make effective use of more sophisticated techniques such as the "snapshot" or its equivalent. He must learn to structure his program to permit meaningful tests to be performed without affecting the desired operation of the program.

Higher level languages

An issue certain to provoke heated discussion among the experts seems to be resolving itself in favor of the use of higher level languages for system programming.

The arguments for higher level languages include:

- Better communication where interaction of programmers is essential.
- More compact documentation—a significant factor in systems of several hundred thousand instructions.
- Facilitation of debugging.
- Closer relation to the conception of the algorithm.
- The ease of transferring the resulting programs, i.e., less machine dependence.
- The potential savings in programming cost resulting from the above.

Meaningful arguments against higher level languages can be made in some special cases

- Real time systems where running time efficiency is paramount.
- Inner loops in large scale computation where computer capability is taxed..
- An adequate compiler for the proposed higher level language is not available.
- Retraining time of experienced assembly language programmers is not commensurate with the schedule.

Only the most sophisticated specialists will claim an advantage for a special system programming language. In time, they may be proven right. (They usually are.)

It should be mentioned that there is considerable exploration of special higher level languages aimed at providing better production tools: specification languages, simulation and modelling languages as well as systems languages.

Never trust the computer manufacturer

This cry is heard with sufficient frequency that it cannot be ignored. At its most vehement, it comes from inexperienced academicians, semi-annually reassured of their omniscience by a sea of bewildered undergraduate faces. Trained in an atmosphere of distrust of their own institutions' administration, they assume contractual responsibility for computer related projects and point with pride to the efficacy of informal arrangements, "cutting through the red tape."

The manufacturer of hardware does not help this situation by responding with a salesman admonished to preserve good relations, but with little or no pertinent technical experience. Properly impressed by the technical knowledge of the young professor who has assured him that only a minor modification is needed, he will make commitments based on his conviction that if it can be done, his company surely will do it. These commitments all too frequently lack the degree of precision required to assess the magnitude of the requested modification and are often made with little or no knowledge of the available resources which would permit a timely response.

Numerous businesses, educational and government organizations succeed in consummating contractual arrangements involving computer software. They do so by availing themselves of the services of personnel experienced in the negotiation of contracts who will ascertain the required detail and level of authorization needed to make a commitment. This suggests that recognition of this problem on the part of responsible members on both sides should be sufficient to remedy this situation.

Documentation—Asset or liability

The importance of documentation in the management of large programming developments is generally accepted. A number of groups have found a formal system of documentation the most effective management tool at their disposal. In its most advanced implementation, such a system of documentation is on-line to a time sharing system available to all participating members of the system programming project.

Difficulties often are related to the programmer's resistance to documentation which may be due to several reasons:

- Lack of tangible evidence of benefit to his own activity.
- The inaccessibility of his colleagues' documentation because of sheer quantity, lack of organization and common format and out of date status.
- Rejection of standards, imposed for reasons he does not appreciate.
- Belief (often confirmed) that he can get along without, and in fact feel at his creative "best" when free to improvise.

Putting a documentation system on-line appears to have overcome this resistance in a manner acceptable to the programmer.

- The system itself can help him by rejecting certain types of inconsistencies.
- He has instant access to the latest version of his colleagues' work.
- Standards have been translated into formatting conventions with which he is familiar.
- He understands that the system must safeguard itself and his programs from unauthorized change. Thus, he more readily accepts the need for authorization to change and implement.

Given such a documentation system, management can institute necessary controls such as a senior programmer's or analyst's approval of a proposed approach prior to implementation. Communication, which is universally identified as a major problem in the development of large programming systems, is facilitated and documentation becomes incidental and concurrent to the development effort. It is conceivable that the management of large programming efforts in the future will be structured in keeping with a well proven system of documentation.

SUMMARY

An attempt has been made to define some of the problems of large scale software developments as seen by managers experienced in this field. This definition has taken the form of identifying generally agreed upon solutions, where they exist, providing the rationale for opposing points of view, and by exploring issues which are largely unresolved.

Among the unresolved issues one finds:

- There is a relative lack of experience at the level of management responsible for setting objectives.
- This is aggravated by the shortage of experts capable of assessing the relevant state of the art.
- In more mature fields of endeavor, managers have been drawn largely from the ranks. Today's programming managers often have a different educational background from the programmers, and are not trained to overcome this difference.
- Complexity, rather than the size of large programming systems has introduced a level of uncertainty by forcing the evaluation of success potential well beyond the design or implementation phase.
- The craft-like nature of programming, i.e., the lack of scientific discipline has proven a real source of problems, such as the difficulty of evolving standards which in turn has made it difficult to specify the task to be performed or to evaluate the end-product.

Issues which have been resolved satisfactorily by at least some organization include:

- Where the advice of experts is available at the outset, it is possible to identify the objectives which should dictate the project organization.
- A project plan can be structured to provide the management tools which allow the measurement of progress against a plan, i.e., means of rendering the development of programming systems visible can be provided.
- Methods of documentation can be developed as an integral part of the effort which aid management in both evaluation and control of the project.

CONCLUSIONS

By committing the time of key executives to this seminar series, organizations large and small have shown their interest in, and support of cooperative efforts to better our understanding of the issues and to share experience.

Some of the organizations represented which have developed effective management techniques in areas other than programming and whose activities span the range from research to production have been able to apply much of their management know-how to programming efforts. The success based upon these techniques has not been unqualified. One reason is the difficulty of relating the visibility of "progress" during a programming effort to ultimate performance. The fact that schedules are being met does not insure success during integration or anticipated performance.

Where management techniques have not evolved and their lack is first felt in the pursuit of a large programming effort, the problems tend to be thought of as unique to programming. Extrapolation and scaling up from past programming experience has proven hazardous. Complexity turns out to be a non-linear attribute.

The management of large programming systems presents some unique challenges. Those most intimately involved recognize the problems. To date, they have had limited success in conveying the full significance of the problem to policy-making management.

To the extent that unresolved problems in the management of large scale software development have been recognized, one can now turn to examining the appropriateness of efforts proposed or under way, as to their potential of providing desired solutions.

REFERENCES

1 C H REYNOLDS
Proc ACM National Conf 1968 334–337
2 D H BRANDON
ibid 332–334
3 E A NELSON
ibid 346–349
4 M E CONWAY
Datamation Vol 14 No 4 April 1968 28–32

APPENDIX I

Participating Seminar Speakers

Mr. Joel Aron
Manager, Boston Programming

IBM
Cambridge, Mass.

Mr. Thomas E. Cheatham
Computer Associates, Inc.
Wakefield, Mass.

Mr. Ted Climis
Director of Programming
IBM
Armonk, New York

Mr. Larry Constantine
Information and Systems Institute, Inc.
Cambridge, Mass.

Professor Fernando J. Corbato
Project MAC
MIT

Mr. Ted Crowley
Bell Telephone Laboratories
Whippany, New Jersey

Dr. Ruth Davis
National Library of Medicine
Bethesda, Maryland

Mr. A. Dean
Manager, Information Laboratory
General Electric
Cambridge, Mass.

Dr. Donald L. Durkey, Vice President
Computing and Software, Inc.
Panorama City, California

Mr. Robert Everett
The MITRE Corporation
Bedford, Mass. and
Baileys Crossroads, Virginia

Professor Jay W. Forrester
Sloan School of Management
MIT

Professor Edward L. Glaser
Director of the Computing Center
Case Western Reserve University
Cleveland, Ohio

Mr. Neil Gorchow
Vice President, Systems Programming
UNIVAC
Philadelphia, Pennsylvania

Mr. William O. Harden
Manager, Data Processing
Union Carbide Corporation
New York, New York

Mr. Alexander S. Lett
Time Sharing Systems Development
IBM
Yorktown Heights, New York

Professor Donald Marquis
Sloan School of Management
MIT

Mr. George A. Mealy
Computer Consultant
Boston, Mass.

Mr. Donald Ream
U.S. Naval Ship Engineering Center
Washington, D.C.

Mr. Carl H. Reynolds, President
Computer Usage Development Corporation
Mount Kisco, New York

Professor Daniel Roos
Director, Systems Lab
Department of Civil Engineering
MIT

Mr. Charles Zraket
The MITRE Corporation
Bedford, Mass. and
Baileys Crossroads, Virginia

Interactive search and retrieval methods using automatic information displays*

by M. E. LESK

Harvard University
Cambridge, Massachusetts

and

G. SALTON

Cornell University
Ithaca, New York

INTRODUCTION

Throughout the world, the design and operation of large-scale information systems has become of concern to an ever-increasing segment of the scientific and professional population. Furthermore, as the amount and complexity of the available information has continued to grow, the use of mechanized or partly mechanized procedures for various information storage and retrieval tasks has also become more widespread. While a number of retrieval systems are already in operation in which the search operations needed to compare the incoming information requests with the stored items are performed automatically, no systematic study has ever been made of the use of man-machine interaction as a part of a mechanized text analysis and information processing system. Specifically, the recent development of high capacity random-access storage mechanisms and conversational input-output consoles should permit a rapid interchange of information between users and system. Such an interchange can then be used to produce improved search formulations, resulting in a more effective retrieval service.

The present report describes and evaluates the performance of a variety of such interactive search and retrieval procedures in which information supplied by the user population is taken into account in an attempt to achieve improved system responses. Several basic approaches to user-system interaction are possible. On the one hand, an attempt can be made to construct refined query formulations, using dictionary displays and similar methods, *before* any file search is actually attempted. On the other hand, an original query can be processed when it is first received, and a query reformulation attempted *after* the results of an initial search are actually available. These two procedures, termed *pre-search* and *post-search*, respectively, can in turn be executed in several different ways: Either the system assumes most of the burden of the query reformulation through an automatic query alteration process, or the users themselves can rephrase their queries using the available automatic displays. In the latter case, the skill of the user population becomes a more important factor. The stored data most important in the pre-search methods might include synonym dictionaries and thesauruses, work frequency statistics, and lists of significant words; the post-search information, on the other hand, consists of the titles, abstracts, or texts of documents retrieved by a previous search process.

The investigation of the various interactive search and retrieval methods is carried out with the help of the automatic SMART document retrieval system.[1,2] The SMART system is a large computer-based retrieval system capable of performing a variety of different text analysis, search, and retrieval operations. Completely automatic text analysis and information searches are made using several different analysis methods and search strategies. Among the main text analysis procedures are synonym recognition, work disambiguation, phrase recognition, statistical term association, and hierarchical text expansion methods.

The effectiveness of the various analysis and search

* This study was supported in part by the National Science Foundation under grants GN-495 and GN-750.

methods may be evaluated by using for this purpose the familiar *recall* and *precision* measures, representing respectively the proportion of relevant material actually retrieved, and the proportion of retrieved material actually relevant. Ideally, all relevant items should be retrieved for the user, while at the same time, all nonrelevant items should be rejected, thus leading to a system where both recall and precision are equal to 1. The performance effectiveness of an operating system can then be estimated by averaging recall and precision figures over many searches and comparing the results with the ideal situation where recall and precision are equal to 1. The SMART system automatically generates for each search a set of recall-precision graphs first introduced by Cleverdon,[3] and also includes procedures for performing computations of the statistical significance of the results. Evaluation data for a wide variety of automatic text processing, search and retrieval methods have previously been published.[4]

In addition to the recall-precision data which reflect the capability of the system to deliver to the user the information he requests, it is also important in an interactive computing environment to take into account the amount of effort required from the user to obtain satisfactory results. Thus, the standard performance of fully-automatic search and retrieval operations must be compared against the improvements obtainable through interactive procedures at additional cost in user effort and computer time.

In the remainder of this study, the effectiveness of various types of interactive search methods is examined, including both pre-search and post-search methods, and semi- or fully-automatic query reformulation procedures. The results are compared using, in each case, the evaluation methods incorporated into the SMART system. Construction principles are then derived for future information services designed to use man-machine interaction during the search process.

Fully-automatic retrieval

In the SMART system, various fully-automatic language analysis procedures are used to normalize the text of incoming search requests and of stored documents. The normalized, reduced forms of the information items, consisting generally of weighted "concept" numbers, are then compared, and the document representations which are most similar to the request representations are extracted from the file as answers to the queries. The language normalization procedures incorporated into the SMART system range from simple word stem matching methods to more sophisticated processes using stored synonym dictionaries and hierarchies, as well as statistical and syntactic analysis methods.[1,2]

Three of the simplest language analysis methods, known, respectively, as *word form*, *word stem*, and *thesaurus* processes may be described as follows:

a. in the word form, or suffix 's', process, no word normalization in the proper sense is used at all, and original words with only the final 's' removed (to confound, for example, "book" and "books") are compared directly;
b. in the word stem method, the original text words are reduced to word stems by a suffix cut-off process to confound words like "analyzer", "analysis", "analyzed", and so on, before the comparison between queries and documents;
c. in the thesaurus process, each word stem is looked up in a synonym dictionary, or thesaurus, where it is replaced by one or more so-called concept numbers, representing synonym classes; the concepts extracted from the thesaurus are then matched instead of the original word forms or word stems.

In all analysis methods, the terms are normally weighted, using word frequency and other criteria, before a comparison is made between stored documents and search requests.

An excerpt from a typical, manually constructed thesaurus is shown in Table 1. Three of the synonym classes defined by the thesaurus mapping are shown in the right-hand side of Table 1. Concept class 346, for example, contains words specifying objects which fly; category 345 lists words associated with weather. If a request were made, asking

"do planes fly when the weather is bad "

the system would retrieve a document stating

"proper meteorological conditions are necessary for the successful piloting of aircraft",

since both document and query would be assigned the concepts 345 and 346.

The handling of ambiguous words in the thesaurus is exemplified by the entry for "wind", which could be either the noun, referring to weather, or the verb, indicating a method of constructing loops or coils. The table shows that "wind" is in two categories, 345 and 233. 345, containing also "weather" and "atmosphere",

Table 1—Thesaurus excerpt

Alphabetic Order			Numeric Order	
Word	Concept Code	Syntax Code	Concept Code	Word
Wide	438	001 043 040	344	obstacle
Will	32032	009 070 043 044 049		target
Wind	345 233	070 043	345	atmosphere
				meteorolog
Winding	233	070 136 137		weather
Wipe	403	043 070		wind
Wire	232 105	070 043	346	aircraft
				airplane
Wire-wound	233	001		bomber
				craft
				helicopter
				missile
				plane

represents the noun, and 233, which contains such words as "winding", "wire-wound", and "solenoid", represents the verbal meaning. Whenever "wind" appears, both 345 and 233 will be entered into the concept vector. Because the word is considered ambiguous, the weight will be divided between these two categories; each will receive half of the weight assigned to "wind".

It should be noted that the thesaurus entries may consist of word stems, so that "meteorolog" suffices to look up "meteorology" and "meteorological". If desired, however, suffixed forms of a word may be entered in the thesaurus; this has been done with "winding", since if only "wind" were in the dictionary, "winding" would also be treated as ambiguous, but the presence of "winding" in the thesaurus makes it possible to identify "winding" in the text with category 233 only.

The high concept number identifies "will" as a so-called common word, not to be used for content identification. The syntax codes shown with the thesaurus entries in Table 1 are not used in the simple automatic thesaurus process.

Since the fully-automatic thesaurus process based on concept number matching is often an effective analysis tool, more sophisticated language normalization methods may not normally be required in an operational retrieval system.

User interaction through pre-search methods

One of the main hopes in obtaining a retrieval performance which goes beyond that presently reached under normal operating conditions is to include the customer in the search process. In particular, fewer errors are likely to be made if the information obtained from the users is not restricted to the search request proper, but is supplemented by a variety of special user indications, or by evaluation data about the acceptability of items previously retrieved by the system in answer to the search requests. User-system interaction is now current for many computer application, often implemented by special input-output console devices, with the help of operating systems which enable the system to render more or less simultaneous service to a large class of users.

In an information retrieval environment, user interaction may take the form of simple dictionary display routines which can be used to present to the user selected dictionary excerpts as an aid in formulating the original search requests, or in reformulating queries which were originally inadequate.[5,6] Alternatively, more sophisticated methods may be used in which the reformulation of the search requests is automatically performed based on feedback information obtained from the user population.[7,8]

The conceptually simpler methods are the *pre-search procedures* which are based on term and dictionary displays of previously stored information. In each case, a user would look at the displayed information and, based on the available data, would decide before any file search is actually attempted how his query could best be reformulated in order properly to reflect his information needs. The following types of pre-search information could be displayed for this purpose:

a. lists of terms included in the user's original search formulation together with word frequency information giving the frequency of use of each word in one or more of the stored document collections;
b. thesaurus excerpts corresponding to the terms included in the user's search formulation, and consisting, for each of the originally available terms, of a complete thesaurus class, including synonyms and other terms related to the original;
c. titles and abstracts of *source documents*, that is, of documents originally known to the user as relevant to his search query (the intent of the user is then to retrieve new documents similar to the source items).

The principal differences between fully-automatic retrieval and retrieval using pre-search interaction are summarized in the flowcharts of Figures 1(a) and 1(b). The pre-search requires the generation of a computer display followed by a manual choice of terms on the part of the user during the query formulation process.

The display of word frequency information is designed to inform the user of the characteristics of the vocabulary which may be used to express his information needs. Thus, if a user notices that many of the terms included in his search request are general terms with a very high frequency of occurrence in the stored document collection (for example, terms such as "computer" and "automatic" in a collection on computer science), he may decide that it is wise to delete these terms from his query so as to prevent the generation of high query-document correlations for many nonrelevant documents. On the other hand, the user may decide to emphasize many highly specific, low-frequency terms by repeating them in the query formulation.

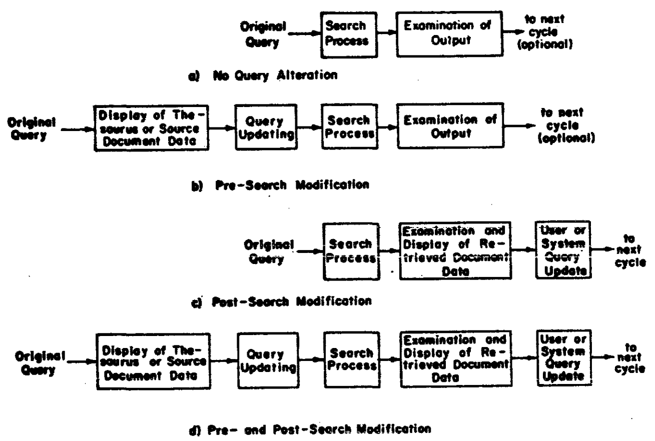

Original Query
Search Process
Examination of Output
to next cycle (optional)
a) No Query Alteration
Original Query
Display of Thesaurus or Source Document Data
Query Updating
Search Process
Examination of Output
to next cycle (optional)
b) Pre-Search Modification
Original Query
Search Process
Examination and Display of Retrieved Document Data
User or System Query Update
to next cycle
c) Post-Search Modification
Original Query
Display of Thesaurus or Source Document Data
Query Updating
Search Process
Examination and Display of Retrieved Document Data
User or System Query Update
to next cycle
d) Pre- and Post-Search Modification

Figure 1—Iterative search procedures

A thesaurus display can be used for *manual* query updating by requesting a printout of the complete thesaurus classes corresponding to each term included in the original query. Consider, as an example, a query dealing with the "contraction of satellite orbits", and assume that the user signifies that he is interested in the "satellite" class. The computer might then type out terms such as

> Discoverer, Sputnik, Vanguard, Cosmos, Moon, rocket, trajectory, countdown, drag, telemetry, etc.

After studying the display, the user might decide that his original query formulation had been insufficiently specific, and the query might then be altered by addition of the terms "Discoverer, Sputnik, Vanguard, Cosmos, drag, and telemetry". The other displayed terms would, however, be rejected as not being germane to the search topic. A second expansion might begin by typing in the term "drag", and then considering the new display of terms related to "drag".

Thesaurus displays are also occasionally useful for the removal from the query formulation of incorrectly used and ambiguous terms. For example, a user interested in information retrieval who identifies his search topic as "IR" might discover that the thesaurus display produces a list of synonyms in the area of "infrared spectroscopy". As a result, the term "IR" would, of course, be removed from the search formulation.

The use of thesaurus displays for manual query updating provides an opportunity for a selective choice of synonym and related terms. That is, the user can choose some terms to be added to the original query, and others to replace already existing ones in an attempt to improve search *precision*. On the other hand, the automatic thesaurus process operates less selectively and provides synonym recognition by the standard process of automatically replacing the word stems originally included in the search requests and documents by the corresponding concept class numbers extracted from the thesaurus. The automatic thesaurus process is thus designed to normalize query and document statements by generalizing the respective formulations rather than by making them more specific. Such a process may be expected to improve *recall*, since more relevant documents could now match the query statements and could thus be retrieved in answer to the respective search requests.

Obviously, the manual query updating method using thesaurus displays places a considerable burden on the user, since he is forced to consider a large number of alternative possibilities before eventually making a move. Moreover, the choice must be made before a search has actually been performed, at a time when he cannot know as yet how well the machine will perform with any potential query formulation.

A comparison of the effectiveness of manual and automatic thesaurus procedures is contained in a later section together with the other evaluation output.

User interaction through post-search methods

The post-search methods are those applicable after an initial search has first been performed. In such a case, one or more documents will already be available, including in particular those items which were initially judged to be most similar to the search requests. These items can now be used in a manner analogous to that previously utilized for the thesaurus displays. Specifically, the titles, or abstracts, of the first few retrieved documents can be examined, and document terms which appear to reflect the wanted subject area can be added to the query statement, while ambiguous and unwanted terms can be removed.

Consider, for example, the previously cited query dealing with the "contraction of satellite orbits", and assume that the first two retrieved items are entitled "Discoverer satellite and South Pacific splash down", and "The moon and the tides". A user could now proceed to add "Discoverer satellite" to the original query, but could avoid the addition of "South Pacific".

The document feedback expansion may be even more difficult to carry out than the dictionary display procedure, since the user is forced to make sophisticated decisions using relatively large text excerpts. Thus, whereas the dictionary display procedure can often be performed in less than a minute per query, approximately four minutes are required on the average for the use of five typical document abstracts. Furthermore, the document expansion process also entails a higher cost in machine time and storage space than the dictionary display, since document abstracts in natural language form constitute a much greater bulk than dictionary excerpts. In addition, an initial retrieval run must first be made before document feedback can be used. On the other hand, a stored dictionary need, of course, not be available for the document feedback method.

Another post-search method is designed particularly for those users who do not wish to assume the burden of query reformulation themselves. For such users, an automatic *relevance feedback* method is available which repuires only a minimum of interaction with the user, since most of the burden is placed on internally stored routines.[7,8,9,10] Specifically, an initial search is first performed for each request received, and a small amount of output consisting of some of the highest scoring documents, is presented to the user. Some of the retrieved output is then examined by the user who identifies each document as being either relevant (R) or not relevant (N) to his purpose. These relevance judgments are later returned to the system, and used automatically to adjust the initial search request in such a way that query terms, or concepts, present in the relevant documents are promoted (by increasing their weight), whereas terms occurring in the documents designated as nonrelevant are similarly demoted.

If the terms from the relevant items are added to the search requests, while terms from nonrelevant items are subtracted, the first query updating operation can be represented by the equation:

$$q_1 = q_0 + \sum_i r_i - \sum_i s_i ,$$

where q_0 is the original query formulation, q_1 is the updated query, r_i is the set of terms identifying the ith document specified as relevant by the user, and s_i is the set of terms identifying the ith nonrelevant document. This process produces an altered search request which may be expected to exhibit greater similarity with the relevant document subset, and greater dissimilarity with the nonrelevant set.

The altered request can next be submitted to the system, and a second search can be performed using the new request formulation. If the system performs as expected, additional relevant material may then be retrieved, or, in any case, the relevant items may produce a greater similarity with the altered request than with the original. The newly retrieved items can again be examined by the user, and new relevance assessments can be used to obtain a second reformulation of the request. This process can be continued over several iterations, until such time as the user is satisfied with the results obtained. Since the method makes very few demands on the user, the automatic relevance feedback process may be expected to be preferred by users unfamiliar with the system operations. On the other hand, the process is not likely to be effective if the user is unable to identify for the system at least one document which is clearly relevant to his needs.

The post-search methods as well as the combined methods making use of pre- as well as post-search information are illustrated in the bottom half of Figure 1. A summarization of all the query updating methods is given in Table 2.

Table 2—Typical query updating methods

Query Alteration Process	Explanation
Pre-Search	
1. Repeated Concepts	User chooses query terms to be repeated for emphasis
2. Thesaurus Display	User chooses terms obtained from thesaurus display to update query (with or without time restrictions)
3. Word Frequency	User looks at display of word frequency information before updating query
4. Source Document	User looks at display of source document before updating
Post-Search	
5. Title Display	User looks at titles of first five retrieved documents before updating
6. Abstract Display	User looks at abstracts of first five retrieved documents
7. Relevance Feedback	Query is updated automatically using relevance judgments supplied by user following an initial search
Combined Methods	
8. Abstract plus Thesaurus	User looks at pre- and post-search information

Evaluation results and discussion

The experimental results included in this section are based on the manipulation of a collection of 200 abstracts of documents in aerodynamics, together with 42 search requests proposed by scientists active in aerodynamics. Complete relevance judgments, prepared by these same scientists, were available which identify for each query the set of relevant documents. The aerodynamics collection has previously been used for test purposes by the Aslib-Cranfield project[3] and by the SMART system.[4]

The thesaurus used for both the manual and automatic query expansion operations contains 3230 word stems and 736 thesaurus classes. This thesaurus was constructed by SMART staff members using text concordances, word frequency lists, standard dictionaries and reference works, and word lists obtained earlier from the Cranfield project. An attempt was made to time the query expansion operations by restricting the use of the thesaurus display to either one minute, two minutes, or more than two minutes. While the output of Table 3 shows that increasingly more terms can be added to the queries as more time becomes available for the updating operations, the differences in retrieval effectiveness are small, and the evaluation output shown represents the output obtained for a display time of two minutes.

The main results are presented first in terms of recall-precision graphs, and then in terms of cost and user effort.

Table 3—Variation in query length

Query Type	Average Number of Significant Terms per Query
Original Query	8.3
Terms added in 1 minute	3.6
in 2 minutes	2.0
later	1.0

A. Recall-Precision Results

The evaluation output is presented in Figures 2 to 7 using the standard recall-precision graphs, averaged in each case over the 42 queries used with the collection of 200 documents. The curves are, as usual, monotonically decreasing, reflecting the fact that as more relevant items are retrieved (as the recall goes up), more irrelevant items are also retrieved (causing the precision to go down). Increasingly more effective retrieval performance is reflected by recall-precision curves close to the upper right-hand corner of the graph where both recall and precision take on ideal values of 1. Next to the graphs, some of the numeric values are presented in terms of recall-precision tables, giving the average precision values at certain selected recall values.

Significance values, computed by a standard t-test, are also included in the output figures, representing in each case the probability that the performance values for two specified processing methods are in fact derived from the same distribution. Thus if the computed probability value is high, the two methods are assumed to be statistically indistinguishable; on the other hand, if the probability value is low—say 0.05 or less—the likelihood that the evaluation results could have been derived from the same data set is very small, and the differences in performance can then be assumed to be statistically significant.

The following principal conclusions can be drawn from the output of Figures 2-7:

a. automatic thesaurus vs. pre-search using thesaurus display (Figure 2):

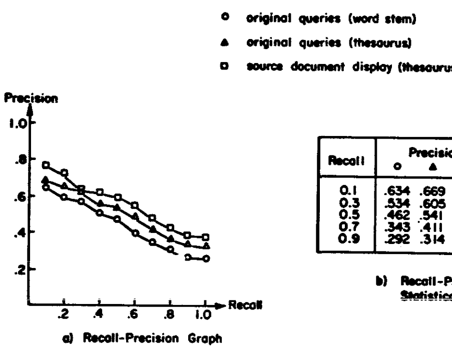
   the automatic thesaurus expansion and the manual expansion using pre-search thesaurus display both produce an improvement in performance over the word stem matching process. Overall, the automatic thesaurus (which requires no user intervention) is superior. At high precision, however, the greater selectivity of the words chosen by the manual process produces better results. The superiority of the automatic thesaurus at medium and high recall is attributed to the previously mentioned difficulty of selecting appropriate terms from the thesaurus display.

b. automatic thesaurus vs. pre-search using source document display (Figure 3):

   the source document display produces a precision improvement of up to ten percent over and above the automatic thesaurus process; however, the table appearing with Figure 3 shows that the improvement is not statistically significant. The relatively modest increase in performance may be due to the fact that the source documents and queries used in the experiment originated with the same authors, so that the source documents contain many of the terms already included in the query statements; also, some of the source documents appear only marginally relevant to the actual queries; both of the interactive pre-search methods turn out therefore to be not substantially superior to the fully-automatic thesaurus method (except at high precision);

c. post-search procedures using displays of titles or abstracts of previously retrieved items (Figures 4 and 5):

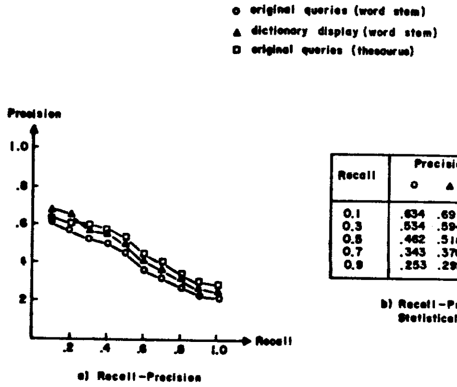
   title and abstract post-search displays are superior to both of the pre-search displays, as shown in Figures 4 and 5. Improvement



Recall	Precision ○	Precision ▲	Precision □	T-test Significance ▲/○	T-test Significance □/▲
0.1	.634	.669	.748	.798	.327
0.3	.534	.605	.603	.339	.869
0.5	.462	.541	.585	.168	.523
0.7	.343	.411	.470	.181	.767
0.9	.292	.314	.362	.252	.770

b) Recall-Precision Tables and Statistical Significance Output

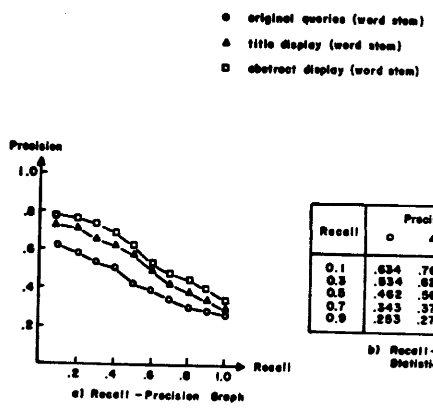
Figure 3—Effectiveness of source document display



Recall	Precision ○	Precision ▲	Precision □	T-test Significance ▲/○	T-test Significance □/○
0.1	.634	.691	.669	.580	.798
0.3	.534	.594	.605	.203	.339
0.5	.462	.510	.541	.199	.168
0.7	.343	.376	.411	.138	.181
0.9	.253	.292	.314	.061	.252

b) Recall-Precision Tables And Statistical Significance Output

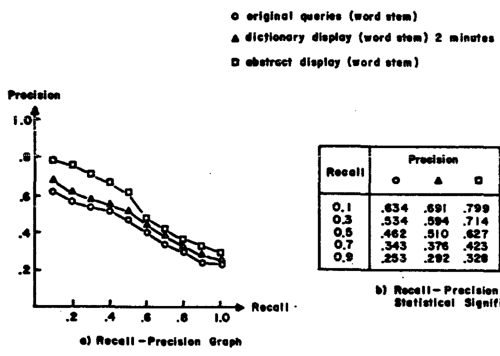
Figure 2—Effectiveness of dictionary display compared with stored thesaurus



Recall	Precision ○	Precision ▲	Precision □	T-test Significance ▲/○	T-test Significance □/○
0.1	.634	.767	.799	.002	.003
0.3	.534	.627	.714	.002	.001
0.5	.462	.584	.627	.007	.001
0.7	.343	.377	.423	014	.001
0.9	.253	.275	.328	.035	.001

b) Recall-Precision Tables And Statistical Significance Output

Figure 4—Comparison of title and abstract display



Recall	Precision ○	Precision ▲	Precision □	T-test Significance ▲/○	T-test Significance □/○
0.1	.634	.691	.799	.580	.003
0.3	.534	.594	.714	.203	.001
0.5	.462	.510	.627	.199	.001
0.7	.343	.376	.423	.138	.001
0.9	.253	.292	.328	.061	.001

b) Recall - Precision Tables And Statistical Significance Output

Figure 5—Comparison of dictionary and text display

with title display is limited to the high precision regions, since the titles are so short that words not in the query are rarely included. The query alterations due to title display are therefore limited to deletion of unnecessary concepts, improving mostly the precision. Abstract displays produce both precision and recall improvements, at the cost of greatly increased work on the user's part. The amount of text examined during an abstract display process is about 1000 words, from which five to ten may be selected for query expansion.

d. automatic thesaurus vs. post-search updating using abstract display and relevant feedback (Figure 6):

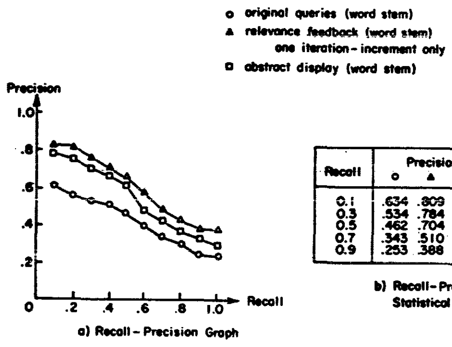
both the manual post-search method with abstract display and the automatic relevance feedback process are superior to the standard automatic thesaurus expansion; the abstract display is best in the very high recall and high precision ranges. The performance differences between the two post-search methods are not significant, although the improvements obtained with both methods over the standard thesaurus process are significant. The relevance feedback output included in Figure 6 is obtained by retrieving, in each case, five documents at a time, asking the user to identify any relevant items, and adding the corresponding terms to the search request.

e. combined pre-search dictionary and post-search abstract display (Figure 7):

Figure 7 shows that a combination of abstracts and thesaurus displays offers an overall improvement of about twenty percent over the standard word stem process, and of ten to fifteen percent over the thesaurus process; in both cases, the improvement is statistically significant. When word frequency information is added to the display, a further improvement results for the word stem procedure, since the user can now ensure that all parts of the query are properly weighted. The output of Figure 7 is approximately equivalent to the automatic relevance feedback process (Figure 6); however, the combined pre- and post-search process requires much more user effort and experience than the relevance feedback method before it can operate successfully.

B. Overall Evaluation

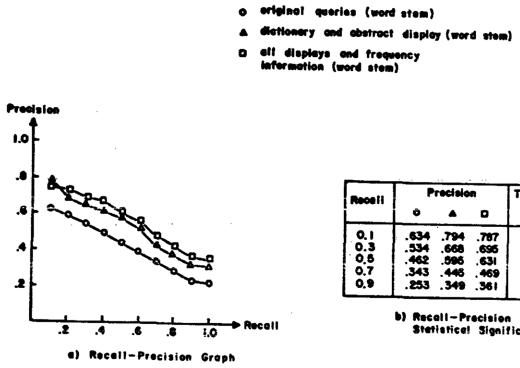
The performance of the various interactive procedures is summarized in Table 4. The first column reflects



Recall	Precision ○	Precision ▲	Precision □	T-test Significance ▲/□	T-test Significance □/○
0.1	.634	.809	.799	.526	.003
0.3	.534	.784	.714	.021	.001
0.5	.462	.704	.627	.077	.001
0.7	.343	.510	.423	.042	.001
0.9	.253	.388	.328	.031	.001

b) Recall-Precision Tables and Statistical Significance Output

Figure 6—Comparison of abstract display with relevance feedback



Recall	Precision ○	Precision ▲	Precision □	T-test Significance ▲/○	T-test Significance □/○
0.1	.634	.794	.787	.009	.009
0.3	.534	.668	.695	.007	.002
0.5	.462	.595	.631	.014	.005
0.7	.343	.445	.469	.001	.001
0.9	.253	.349	.361	.002	.002

b) Recall - Precision Tables And Statistical Significance Output

Figure 7—Comparison of combined methods (word stem process)

Table 4—Performance summary

Processing Method	Demands on Computer	Demands on User	Precision Improvement Over Word Stem Match	
			Low Recall	High Recall
A. *Fully Automatic*				
word stem match	normal	none	—	—
automatic thesaurus	normal	none	+4%	+6%
B. *Pre-Search Interaction*				
thesaurus display	normal +	medium-high	+6%	+4%
source document display	normal +	medium	+8%	+5%
C. *Post-Search Interaction*				
title display	high	medium	+13%	+2%
abstract display	high	very high	+17%	+7%
relevance feedback	high	low	+10%	+7%

computer demands; the second, user effort; and the last two reflect search effectiveness in terms of recall and precision improvements over and above the normal word stem matching method.

Since the post-search methods require two separate file searching operations—one prior to the interactive process and one following it—the computer demands are comparatively higher for post-search than for the other methods. Thus, when search time may be expected to be considerable—for example, for very large collection sizes—the pre-search procedures may become mandatory.

From the user's viewpoint, the less information is displayed, the easier will normally be the interactive process. Thus, the relevance feedback procedure is simplest, since the user must merely identify one or another document as either relevant or nonrelevant; the pre-search thesaurus displays, and the post-search abstract displays are hardest, since complicated decisions are required to update the search requests.

Turning now to the performance parameters, it is seen in Table 4 that, everything else being equal, the post-search methods are more powerful than the pre-search procedures. (Unfortunately, those are also the methods which put the highest demands on the computer.) One obvious reason why the post-search methods operate more reliably is that a computer search has already been performed before the user is asked to update the query. Thus, the query alteration process is undertaken with prior knowledge of how well the original query has performed. The post-search alteration can then be used to initiate small changes for queries requiring only little improvement, and more massive changes for the others. For the pre-search methods, no such prior information is available.

Of the post-search methods, the best performance is obtained with abstract display; however, this method also makes the greatest demands on the user. The relevance feedback method is not much inferior and more preferable from the user's viewpoint.

To summarize the performance and cost indications, the following search strategy would appear to be useful under most circumstances:

a. normally, use standard automatic thesaurus method without user interaction;
b. if improvement is needed and search time is not excessive, use relevance feedback;
c. if search time is at a premium, use pre-search source document or thesaurus display;
d. on the other hand, if high retrieval performance is mandatory, try post-search abstract display.

The difficulties of the manual query updating methods may be illustrated by the example of query 317, reproduced in Table 5. The original word stem retrieval run produces the two relevant documents in rank positions 4 and 10. From the thesaurus display, the following words were selected: "elastic", "resilient", "unstiffened", "modulus", "aeroelastic", "laminar-boundary-layer". This promotes the two relevant documents to rank positions 2 and 5; however, the automatic thesaurus run yields rank positions 2 and 4, without any user interaction. When the post-search displays are used, the results are similar. Title display is not very

Table 5—Typical manual query updating

Query 317: Has anyone investigated theoretically whether surface flexibility can stabilize a laminar boundary layer?

(Two Relevant Documents)

Processing Method	Terms "Added" or *Deleted*	Ranks of Relevant Documents
A. *Fully Automatic*		
word stem match	—	4,10
automatic thesaurus	—	2,4
B. *Improved Searches*		
word stem plus thesaurus display (pre-search)	"unstiffened", "modulus", "elastic", "resilient", "aeroelastic"	2,5
word stem plus title display (post-search)	"theoretical"	4,9
word stem plus abstract display (post-search)	"elastic", "resilient", "theoretical"	1,6
C. *Perfect Searches*		
automatic thesaurus plus relevance feedback	—	1,2
word stem plus abstract display plus automatic thesaurus	"elastic," "resilient"	1,2
thesaurus display plus abstract display plus word frequency display	*anyone*, *investigate*, "theoretical", "flexibility", "analytic", "resilient", "calculate", "unstiffened", "aeroelastic", "laminar-boundary", "flexure", "elastic".	1,2

effective for this particular query, yielding only an indication that "theoretical" should be increased in weight, which raises the rank positions of the relevant from 4 and 10 (in the original word stem run) to 4 and 9. Abstract display is more fruitful, adding "elastic" and "resilient" as well. This increases the ranks of the relevant documents to 1 and 6. However, the same query, now processed through the automatic thesaurus (abstract display and automatic thesaurus run) yields perfect performance, as does the automatic thesaurus run with relevance feedback.

To achieve perfect performance using only manual updating methods and word stem matches, it is necessary to utilize a combined thesaurus display, abstract display, and word-frequency information, which yields the following rather complex set of changes: delete "anyone" and "investigate"; increase the weight on "theoretical" and "flexibility" by a factor of two; add with weight of one the word "analytic", "resilient", "calculate", "unstiffened", "aeroelastic", and "laminar-boundary"; add with weight of two "flexure"; and add with weight of three "elastic". These changes produce a word stem run with perfect performance, but at far greater time and trouble than the automatic thesaurus with abstract display run. The exact adjustment of the term weights is normally performed more accurately and more easily by the automatic thesaurus. The manual methods are thus best reserved for users with the skill and interest to consult lengthy displays and to make complex decisions.

A meaningful cost analysis is difficult to make without the use of an operational time sharing system to

Table 6—Estimated cost figures

Processing Method	Estimated Cost per Query 50,000 documents	Estimated Cost per Query 100,000 documents
A. *Fully Automatic*		
word stem match	\$ 5.00	\$10.00
automatic thesaurus	\$ 5.00	\$10.00
B. *Interactive Pre-Search*		
thesaurus display	\$ 6.00	\$11.00
source document display	\$ 5.50	\$10.50
C. *Interactive Post-Search*		
title display	\$10.50	\$20.50
abstract display	\$13.00	\$23.00
relevance feedback	\$10.50	\$20.50
D. *Partial Search*		
cluster searches (one-tenth of collection)	\$ 0.50	\$ 1.00
cluster search plus relevance feedback	\$ 6.00	\$11.50
cluster search plus abstract display	\$ 8.50	\$14.00

Assumptions: machine cost \$75.00/hour
document scan 5msec/doc
central processing cost 0
human time \$10.00/hour

perform the experiments. Table 6 contains an estimated cost summary based on running times for the IBM 7094. Machine and user costs are assumed to be \$75.00 and \$10.00 per hour, respectively. Scanning time is five milliseconds per document, and additional central processor time is ignored. Table 6 shows that the post-search methods are clearly the most expensive (they also are the most effective), with relevance feedback relatively cheaper than abstract displays. In general, the automatic procedures appear economically and operationally better suited to the retrieval operations than the manual methods. Since the cost of human time may be expected to continue to increase relative to the cost of machine time, the automatic procedures may grow even more attractive in the future.

The bottom part of Table 6 shows that processing cost goes down drastically if *partial* searches of the collection are performed, rather than full searches. Such partial "cluster" searches are implemented with the SMART system; however, the cluster searches cannot be used if a recall performance higher than about 50 percent is required.[11]

CONCLUSION

The best overall process for precision purposes is the abstract display used in conjunction with a word stem matching procedure. For recall purposes, a combination of abstract display with thesaurus word normalization appears best. The automatic relevance feedback approximates the abstract display method while requiring much less user effort. Considering the complexity of the abstract display system, a sensible set of recommendations for high performance real-time retrieval would be the following:

a. for highest precision, use title display and word stem matching;
b. for highest recall with normal users, use the automatic thesaurus followed by automatic relevance feedback; with experienced and patient users, use abstract display and dictionary display plus frequency information;
c. for maximum cost reduction at lower per-

formance, use partial searches of the document collection.

These rules provide a graded set of feedback methods, ranging from automatic procedures which make only minimal demands on the user and are suitable for novices (automatic thesaurus expansion, relevance feedback), to methods permitting sophisticated user-system interaction which combine the best features of manual and automatic query adjustment (thesaurus and abstract display). One may expect that a suitable mix of user feedback procedures can be found to produce optimal retrieval under many different conditions over many types of user classes.

REFERENCES

1 G SALTON M E LESK
The SMART automatic document retrieval system—an illustration
C ACM Vol 8 No 6 June 1965

2 G SALTON ET AL
Scientific reports on the SMART system to the National Science Foundation Nos ISR-11 ISR-12 ISR-13
Department of Computer Science Cornell University Ithaca New York June 1966 June 1967 and January 1968

3 C W CLEVERDON J MILLS E M KEEN
Factors determining the performance of indexing systems
Vol 1 Design Aslib-Cranfield Research Project Cranfield College of Aeronautics 1966

4 G SALTON M E LESK
Computer evaluation of indexing and text processing
J ACM Vol 15 No 1 January 1968

5 R M CURTICE V ROSENBERG
Optimizing retrieval results with man-machine interaction
Center for the Information Sciences Report Lehigh University Bethlehem Pa 1965

6 H BORKO
Utilization of on-line interactive displays
In Information Systems Science and Technology, D Walker editor Thompson Book Co Washington D C 1967

7 J J ROCCHIO JR
Document retrieval systems—optimization and evaluation
Harvard University Doctoral Thesis Scientific Report No ISR-10 to the National Science Foundation Harvard Computation Laboratory March 1966

8 J J ROCCHIO G SALTON
Information search optimization and iterative retrieval techniques
Proc F J C C Vol 27 Spartan Books November 1965

9 G SALTON
Search and retrieval experiments in real-time information retrieval
Proc IFIP Congress—68 Edinburgh August 1968

10 E IDE
User interaction with an automated information retrieval system
Scientific Report No ISR-12 to the National Science Foundation Sec VIII Department of Computer Science Cornell University June 1967

11 R T GRAUER M MESSIER
An evaluation of Rocchio's clustering algorithm
Scientific Report No ISR-12 to the National Science Foundation Sec VI Cornell University Department of Computer Science June 1967

The LEADER retrieval system

by DONALD J. HILLMAN and ANDREW J. KASARDA

Lehigh University
Bethlehem, Pennsylvania

INTRODUCTION

The LEADER system is a new service-oriented prototype designed to meet the retrieval needs of research scientists working within or in conjunction with the Center for the Information Sciences at Lehigh University. In the first part of this paper, we describe the major conceptual apparatus and principal design features of LEADER, while the second part contains a brief discussion of system implementation and user interaction.

The name "LEADER" is an acronym for "**LE**high **A**utomatic **D**evice for **E**fficient **R**etrieval," and is thus similar to other acronyms in possessing both an intended meaning as well as an actual referent. Imaginative readers can undoubtedly supply alternative and presumably more ribald interpretations of the same six characters, but this is rather incidental to the main goal of the LEADER system, which is to provide a very highly user-oriented facility for the negotiation of open-ended inquiries and interactive browsing. To help meet this objective, the system includes on-line processing of requests, using a novel and relatively inexpensive hardware configuration, and serially organizes its output in the form of document references, citations to documents, and complete textual passages selected from one or several documents, in any way that the user specifies. This ability of the user to control output is but one feature of an overall interactive procedure which begins when an initial request is entered into the LEADER system in the form of a set of sentences describing the user's problem. Each input sentence must, of course, be grammatically well-formed, but there is no restriction on vocabulary. A typical inquiry might read:

> "I would like to know whether modular bounded functionals have ever been used in theoretical studies of retrievable sets, and if so by whom and with what results. If there has been no application of this type, I would be interested to learn of any work in retrieval theory that makes use of Borel functions. If there is no such work, please direct me to retrieval studies involving topological measures or metric spaces in general."

Inquiries such as these are presented directly to the system and displayed on a CRT scope. As each inquiry is displayed, it is also automatically analyzed by the same procedures used to process the full text of input documents. That is to say, LEADER treats both documents and queries as entities of the same logical type to begin with, so that the logical and referential structure of an inquiry is accorded just as much importance as the structure of a document. The goal of text processing is therefore the same throughout, *viz.*, to determine what each group of input sentences is *about*, whether they constitute a document or an inquiry, and to establish major patterns of conceptual relatedness between documents and terms used either in document or query characterization. The text-processing features of LEADER thus include elements of syntax, semantics, and logic.

After the sentences of an initial inquiry have been analyzed into concept-denoting expressions and their logical interrelationships, LEADER is able to fashion an appropriate response to the user's retrieval needs by comparing the conceptual structure of the inquiry with the general structure of the data base. This comparison is conducted *via* a man/machine dialogue in which LEADER instructs and interrogates the user, attempting to acquaint him with the nature of its stored information so that each inquiry can be negotiated through successive modifications of the user's stated interests. The dialogue itself is carried out on a CRT scope.

The user may call for document references, citations, or passages of relevant text at any time during the negotiation, so that by a process of selective browsing he may assist LEADER to arrive at the most appropriate solution to his retrieval problem. What the user wants in the way of final output is a matter for him to decide. In most cases, users prefer to read portions of the text of selected documents on the CRT scope, and to ask for hard copy of what they state to be the most interesting or pertinent of the documents that have been displayed for them on the CRT. Such hard copy is provided by either a line printer or a low-speed terminal.

Work on the theoretical basis of LEADER was begun in 1962, and preliminary implementation of the theory took place in 1965-1966, when programs for full-text analysis were written and tested. In 1966-67, system components for connectivity and decision were developed, and a small scale prototype went into operation in the summer of 1967. A full scale version of LEADER is now virtually complete, employing a data base of technical articles and reports in information science, the full text of which is in machine-readable form.

The design features of the system call for a substantially different approach to evaluation, largely because such widely-used and familiar notions as "precision" and "recall" have no operational meaning for LEADER. For this reason, we have proposed new methods for evaluating system performance, working directly with a real user population composed of Lehigh faculty and staff. These methods are now under development.

A full discussion of the text-processing procedures now being used in LEADER is contained in earlier publications.[1] For ease of reference, however, we shall begin our detailed description of system design and operation with a brief synopsis of theory and software development up to the present time.

Retrieval theory

The theoretical basis of LEADER may be described as a deductive framework for the operations governing the retrieval of those messages whose content is such as to make them measurably relevant to negotiable inquiries. By "message" we mean a well-formed assemblage of lexical items constituting a sentence, a document, a portion of a document, a set of sentences culled from several different documents, a document description, a bibliographic reference, a citation, and so on. This liberal interpretation of "message" is intended to reflect the wide variety of useful retrieval responses to (a) different types of inquiry, and (b) different stages within the negotiation of a given inquiry. It is thus apparent that the LEADER system is designed to incorporate within a single framework the hitherto separate functions of document retrieval, data retrieval, reference retrieval, *etc.* There are numerous reasons for combining these different functions into one integrated mechanism, some of which relate to improved hardware and software, while others arise from changes in design philosophy. By far the most important reason, however, is that inquiries put to a retrieval system are not all of the same kind, so that it is imperative to build into the system as many different types of response as there are different kinds of inquiry. Many requests, for example, require a substantial degree of retrieval completeness, while others may be met by rather simple lists of document references. A flexible response capability has therefore been programmed for LEADER, permitting the system to vary its several outputs as the user specifies, whether during the course of negotiating a single inquiry, or responding to different types of request.

The theoretical framework of LEADER is described in a series of publications[2] dating from 1962. At its most recent stage of evolution, the theory defines three major groups of operations leading to a corresponding identification of three main system components, each with its own sub-theory. We distinguish a text-processing, a connectivity, and a decision component, and describe below how their theoretical foundations were established.

Text-processing subtheory

This subtheory formalizes a set of procedures for assigning content-indicating symbols to documents. In the LEADER system such symbols are termed "characteristics", while the activity of assigning characteristics to documents is called "characterization".

Since we characterize a document in order to mention the topics it deals with, it is clear that the objective of text-processing is to enumerate a set of topic-denoting expressions for every document in the collection. By limiting our scrutiny to noun-phrases, it is possible to identify topic-denoting expressions as those substantival expressions occurring as *arguments* of logical relations. For example, the sentence "The four-group constitutes a subgroup of the tetrahedral group" expresses a binary relation whose arguments are "the four-group" and "the tetrahedral group". We say that these two noun-phrases are topic-denoting expressions, or that the sentence deals with the topics of the four-group and the tetrahedral group.

In the complete text-processing theory of LEADER, each document is regarded as a complex of ordered sentences, every one of which must be reduced to its underlying logical relations. We say that any sentence expressing a relation has *canonical form*, or is a *canonical component*, and call the process of reducing an

English sentence to its canonical components *canonical decomposition.* An algorithm has been written to identify the canonical components of every input sentence of a document, and to isolate all noun-phrases occurring as arguments of such components. These noun-phrases are potential document characteristics whose actual selection is subject to rules described in another section.

Connectivity subtheory

The second subtheory is concerned with (1) a class of operations defined on characteristics to establish their interconnections, and (2) a class of operations defined on documents and their characteristics. The former class of operations serves to formalize the concept of *association*, here interpreted as the calculable relatedness of topic-denoting characteristics. The latter class of operations formalizes the notion of *affiliation* between characteristics and documents. This concept is used to measure the strength of the connection between a given characteristic and any document to which it has been assigned.

Because the output of text-processing functions as the input data for connectivity, it is necessary to provide a link between the respective subtheories. This can be accomplished by stipulating that terms are connected at the first level if they occur as the arguments of a relation.

A criterion was established for measuring the degree of connection between each term used to characterize a document and the document itself. All characteristics are assigned *weights* relative to their documents, represented by the entries of a chaiacteristic by document matrix. Multiplication of this matrix by its transpose yields a characteristic by characteristic matrix defining connections between characteristics *via* one document.

This matrix is partitioned into submatrices defining *genera.* A genus is defined as a connected component of a *graph* of characteristics, and is therefore made up of characteristics every pair of which is connected by a sequence of edges in the graph.

Each genus submatrix is converted to a transition *matrix*, and it is shown that properties of connectivity within general have an appropriate model in the theory of ergodic Markov chains.

A description of the procedures used to establish connections is provided in a later section.

Decision subtheory

The goal here is to construct a theory of the operations governing the deductive and associative liaisons between document representations and inquiries. This particular theory has gone through many stages, and is still in fact being rather vigorously studied. A major reason for continued investigation is that our conception of the decision component had been revised several times in response to the increasing possibilities of ever more sophisticated retrieval. Thus, starting with the relatively straightforward problem of constructing a model for subject document retrieval, we have gone on to examine associative models, and have finally arrived at the stage whereby the retrieval process is defined as an interactive procedure between the user and the system in which there are many different levels and types of response, ranging from a simple enumeration of bibliographic information, at the lowest level, to a full negotiation of a complex query involving browsing, text-display, and question-answering, at the highest level. As the decision component has become more complex and sophisticated, so its corresponding subtheory has passed through many stages of mathematical development, the most significant of which have been described in earlier publications.[3]

System implementation

Hardware configuration

The LEADER system has been implemented on an IBM 1800 Process Control computer. The process controller (cpu) has a 16K, 4μ second main core memory and a 2310 disk controller with two 2315 disk cartridges that provide over a million words of on-line peripheral storage. An IBM 1442 card reader/punch, an 1816 typewriter console and a 2260 cathode ray tube terminal are used for basic system I/O operations.

The IBM 1800 Process Control system operates under TSX, an IBM-written time-sharing executive with real time capability. It is through TSX that LEADER functions.

Users can communicate with the LEADER System in either of two ways. The first method is *via* an 1816 typewriter-like terminal that transmits data at about 15 characters per second over the 1800's standard data channel. It is an on-site (local) device. The second method is *via* a 2260/2848 Video Display-Controller which transmits data at 240 characters per second over a selector channel. The 2260 CRT is a video display terminal with a 960 character display buffer (screen) and a typewriter-like console for data entry. Up to eight 2260 CRTs can be attached to the 2848 display controller at various remote locations.

Software

The LEADER System software can be divided into three main categories based on producer and function. These are:

IBM supplied system software
LEADER System interface software
LEADER System operational software

The IBM supplied software consists of the standard software packages available with the 1800 Process Control system.

The LEADER System interface software performs two basic functions. The first is a system interface routine that links TSK and the 2260/2848 display controller. It services hardware interrupts generated in the LEADER System's operational environment. The second is a program interrupt-servicing routine that supervises all LEADER operations requested by the user.

The LEADER System operational software consists of three classes of routines designed to perform the functions required by each of its three components. They are:

Text Processing Component Program Package
  Text Entry Compiler (LETEXT)
  String Manipulation Compiler (LECOM-II)
  Syntactic Analyzer (LEGRAM)

Connectivity Component Program Package
  Connectivity Operations Supervisor
  Connectivity Matrix Routines
  Retrieval File Generation Routines

Decision Component Program Package
  Request Analyzer
  Request Negotiator
  Display Generator

Data base

The Lehigh University Center for the Information Sciences maintains a literature collection in information science and engineering as both an experimental data base and an active reference source for faculty and students. At present there are approximately 3000 documents in the collection, and our intent is to limit the size to 10,000 items. However, we wish to make the collection highly dynamic so that is will continue to be substantively useful and meaningful to researchers and students. This will be accomplished by input filtering and item deletion. At the input stage, only high-quality documents dealing with topics of substantial interest will be accepted for processing and inclusion. The criteria for selection are as follows.

1. General papers on information retrieval, documentation, and computer appreciation are excluded, unless there is a specific reason for acceptance, such as basic policy statement (e.g., Weinberg Report) or a particularly cogent description of the field, or a general paper in which specific important data are included.
2. The following areas are included in the collection, with particular emphasis on research, experimentation and systems analysis.

   (i) Automatic indexing and abstracting;
   (ii) Syntactic analysis (but not when the orientation is exclusively mechanical translation);
   (iii) Logical and mathematical studies of retrieval, relevance, indexing, etc.;
   (iv) Basic systems studies, including costs, major system studies, parallel system problems, compatibility, library automation;
   (v) Behavioral studies of users, questions, effect of information on management decision, the research program, and on engineering processes.
   (vi) Programming languages, particularly symbol manipulation languages;
   (vii) Pertinent reviews and bibliographies;
   (viii) Peripheral material considered pertinent; automata studies, mechanical translation, self-organizing systems, cognitive processes, neurophysiology, linguistics;
   (ix) Education of personnel in information handling and the information sciences.

With respect to item deletion, user reaction is the basis for upgrading the quality of the collection. Since it is hardly meaningful to experiment with documents that real users find of little interest, we shall delete items found to be of minimal value, and replace them by documents suggested by users.

These filtering, deletion and replacement operations are not only necessary for quality control, but ensure that the experimental collection is non-static. Collections that are sealed off against new entries misrepresent real-world conditions, and our approach to system design is to deal exclusively with dynamic collections, whose membership is subject to continuous modification.

Since the literature collection is, to the best of our knowledge, the only formal and accessible set of high-quality documents devoted exclusively to information science and engineering, it has certain unique features. Not only are faculty, research personnel and students using the collection for experimentation, but the literature itself is of substantial interest to them. In order to make the literature available for reference purposes, it is controlled by conventional library methods. All incoming documents are classified and shelved, and author and subject indexes are maintained.

Text entry and processing

Each document selected for inclusion in LEADER is entered in its full-text version. The instrument for such text-entry operations is a data-editing, non-numeric, and non-interpretive compiler known as LETEXT (see section on *Software*).

Text is entered by direct keying from an 1816 terminal or a 2260 CRT and stored on disk files. Input creates both visual and computer records, and extensive editing and error correction features are available.

Text entry and editing in LETEXT are background jobs on the IBM 1800 time-shared computer, so that the machine-readable data base of LEADER may be continuously augmented while retrieval operations are in progress.

The procedure for analyzing the sentential structure of documents entered into LEADER involves three distinct steps. The first is a dictionary look-up program for identifying the key functor words and phrases of any sentence. The dictionary currently contains approximately 2,000 entries, including such items as suffixes, prepositions, prepositional phrases, conjunctions, conjunctive phrases, auxiliary verbs, and so. When found, each functor word is replaced by a specially coded syntactic category symbol.

The second program assigns syntactic category symbols to the remaining words of the sentence on the basis of a context-sensitive analytic grammar.

The third program reduces the strings of categories produced by the first two programs to their canonical components. These, as previously explained, are strings of categories having the form of a sentence expressing a logical relation.

To illustrate the procedure down to the decomposition stage, consider the following example of an input sentence:

> Brown's algorithm is a generalization of Euclid's algorithm capable of treating three or more equations in one process, but nevertheless introduces many extraneous factors. (1)

The output of the dictionary look-up procedure for this sentence is:

> Brown's *A* algorithm *U* is *V* a *ART* generalization *N/A* of *P* Euclid's *A* algorithm *U* capable *A* of *P* treating *VL* three *A* or *P* more *A* equations *N/A* in *P* one *A* process *N/V*, *CC* but *C* nevertheless *C* introduces *VX* many *A* extraneous *A* factors *N/A* (2)

where the category symbols stand for the following:

A	adjective
U	unknown
V	verb
ART	article
N/A	noun or adjective
P	preposition
VL	present participle or gerund of unknown verb
N/V	noun or verb
CC	comma
C	conjunction
VX	present or past tense of transitive or intransitive verb.

The second program replaces the unknowns of (2) with calculated category symbols and resolves multiple category assignments such as *N/A*. We have:

> Brown's *A* algorithm *N* is *V* a *ART* generalization *N* of *P* Euclid's *A* algorithm *N* capable *A* of *P* treating *VL* three *A* or *P* more *A* equations *N* in *P* one *A* process *N*, *CC* but *C* nevertheless *C* introduces *VX* many *A* extraneous A factors N (3)

The third program decomposes (3) into the following canonical components:

(i) Brown's *A* algorithm *N* is *V* a *ART* generalization *N* of *P* Euclid's *A* algorithm *N*

(ii) Brown's *A* algorithm *N* is V capable *A* of *P* treating *VL* three *A* or *P* more *A* equations *W* in *P* one *A* process *N*

(iii) Brown's *A* algorithm *N* introduces *VX* many *A* extraneous *A* factors *N* (4)

This example shows clearly how the phrase "Brown's algorithm" has three *logical* occurrences, although it appears just once in the original sentence.

When every sentence of an input document has been reduced to its canonical components, another program selects document characteristics from such components and assigns numerical weights to selected characteristics. Since all potential document characteristics are noun-phrases occurring as arguments of the relations expressed by canonical components, they are extremely easy to identify.

Each selected characteristic occurring as an argument of an n-termed relation R is said to have n lines of connection to R. A characteristic t is said to have m lines to connection to a document D if m is the sum of all lines of connection between t and the relations of D in which t appears as an argument.

The weight of a characteristic relative to a document is the sum of its lines of connection to the document. Thus, if a characteristic t occurs once as the argument of a two-termed relation in a given document D, and at another time as the argument of a three-termed relation of D, the weight of t relative to D will be five.

In support of this measure, it is essential to realize that any document is a coherent complex of assertions. Its function is not to enumerate haphazard and disconnected pieces of information, but to give an organized account of its subject matter. That is, a document fits together concepts embodying knowledge of a field of inquiry. The connectivity of terms *via* predicates contributes greatly to a document's coherence. Such connectivity is, in fact, basic to all other types of connectivity.

In summary, the procedure operates on the full text of documents written in technical English, reduces each text sentence to a string of syntactic categories, resolves each category-string into the set of its canonical substrings, identifies potential document characteristics within the canonical substrings, and finally assigns a weight to each characteristic relative to its parent document.

Once a document has been processed by LEADER, it is analyzed on a sentence-by-sentence basis for characterization and connectivity decisions. That is, the structure of each sentence is explored to determine which of its noun phrases qualify as potential document characteristics. The output of this analysis is a set of weighted source-derived noun phrases that occurred in referential positions within their respective sentences. The phrases are sorted, merged and then entered into a term-document affiliation matrix. By performing appropriate matrix operations on the affiliation matrix these noun phrases may be grouped into distinct sets of mutually related terms known as "genera." Further processing of these genera can be done to remove low-entropy terms, hence further refining the quality of the document characteristics.

Files

The corpus actually used in the LEADER System consists of approximately 1000 documents, each of which is analyzed by the fully automatic text processing and connectivity procedures described above. The text of every document is reduced to its major assertions, *i.e.*, sentences making assertions about the topics denoted by the document's characteristics. Such sentences may be grouped by function, if desired. For example, descriptions of results form one group, while descriptions of experimental procedures form another. It can easily be shown that the procedures of canonical decomposition assist in defining such groups. The ability to identify and retrieve such sentences is, therefore, an important feature that has been incorporated in the LEADER System.

The analyzed and reduced text of each document is placed on the IBM 1800's disk storage cartridges along with the files required for retrieval operations on the domain of documents, their respective analyzed texts, their characteristics, and the connections between characteristics. These files contain bibliographic data on documents, including the storage locations of the individual sentences of their text, and describe the nature and strength of the connections among characteristics, components of characteristics, genera of characteristics, and documents.

To generate the retrieval files, we make use of the matrices produced by the connectivity procedures described above. Output from the text processing operations is in the form of an alphabetically sorted set of source derived noun phrases with a record format as shown in Figure 1.

Document Number	Source Derived Phrase	Weight of Phrase Relative to Document	Sentence Numbers in which Phrase occurs

Figure 1—Text processing output record format

These phrases are used to generate the term-document matrix on the one hand, and to build the Source Derived Phrase Dictionary. Its record format is similar to that given in Figure 1, except that identical phrases are merged relative to a given document. See Figure 2.

Phrase Number	Source Derived Phrase	Document Number	Weight of Phrase Relative to Document	Sentence Numbers in which Phrase Occur	Doc. No.	Wt.	Sent. Nos.

Figure 2—Source derived phrase dictionary record format

From this file and information contained in the partitioned term-term matrix, we obtain the Noun Phrase-Word Profile Dictionary. Each phrase is reduced to its non-trivial word components. Along with each such word, the phrase number of the phrase from which the word was obtained and the genus number of that phrase are combined to for a Temporary Word File with records as shown below.

Word	Genus Number	Phrase Number

Figure 3—Temporary word file

These records are then sorted first alphabetically by word, next by genus number, and finally by phrase number. Then this sorted list is merged into the Noun Phrase-Word Profile Dictionary whose record format is shown in Figure 4.

Word	Genus Number	Phrase Numbers	Genus Number	Phrase Numbers	. . .

Figure 4—Noun phrase-word profile dictionary

The Temporary File can then be discarded. Finally, we generate the Phrase Affiliation File which is derived directly from the partitioned term-term matrix. It contains the genus number, phrase number and affiliation value.

The genus number is assigned to each of the submatrices in the partitioned submatrix. Within each genus (submatrix), the column and row entry of the submatrix is a source-derived phrase, while the component entry within the submatrix is the term-term affiliation value derived for each pair of terms within the given genus. The result of merging this information is the Phrase Affiliation File. See Figure 5.

Genus Number	Phrase Number	Affiliated String Number	Affiliation Value	Affiliated String Number	Affiliation Value	...

Figure 5—Phrase affiliation file

The end product of the sorting and merging of the information contained in the matrices are the three main retrieval files described above. They are:

Source Derived Phrase Dictionary
Noun Phrase-Word Profile Dictionary
Phrase Affiliation File

There are four other data files used by LEADER. These are:

Special Topics File
Author/Title Information File
Citation File
Document File

The Special Topics File is a topic-oriented file made up of individual sentences from various associated documents. The remaining files are generated directly at input of the document *via* LETEXT.

Interactive retrieval

LEADER is designed to encourage user interaction with the structured material of a corpus of scientific or managerial data so as to maximize the influence of information flow on decision making. The data entry procedures are sufficiently general to accommodate several different types of data base, provided only that each consist of well-formed English sentences. In addition, the response capability is flexible enough to permit retrieval ranging from the enumeration of simple bibliographic data, on the one hand, to full-text display, on the other.

The extent to which information flow contributes to decision-making is certainly affected by the ability of an information system to adapt itself to a user's needs. It is for this reason that retrospective literature searches are no longer sufficient in information retrieval. It is now necessary to develop the framework and experimental procedures requisite for a true interaction between user and store. Most experimental work to date looks upon both the inquiry and the relevance of answers as single events. We think this is a mistake and that an inquiry is merely a micro-event in a shifting, adaptive process. It is not a command, as in conventional search strategy, but rather a description of an area of doubt in which the question is open-ended, negotiable and dynamic. The immediate goal of the LEADER system is thus to provide a facility that will, within feasible and practical limits, offer the user a range of experimental configurations which he can amend or add to as necessary. Its long range function is to design and test techniques that will allow inquirers to be instructed in the system, to browse, to query, to be interrogated by the system, and to be shown various strategies for search.

To begin a retrieval dialogue, a user enters a preliminary search description (*via* the 2260 CRT console) in the form of a set of declarative English sentences. LEADER's syntactic analyzer then reduces these sentences to a set of noun phrases which were found to be in referential positions within the sentences. Next, the noun phrases are reduced to non-trivial component words. This is done because it is rather unlikely that a noun phrase presented by a user will precisely match any of the noun-phrases in LEADER's source derived Noun Phrase Dictionary. Thus, each component word of each noun phrase derived from the request is looked up in the Noun Phrase-Word Profile Dictionary to determine its respective genus association and phrase affiliation, if any exists. The results are then merged by genus into maximal sets of ranked noun phrases, if any exist, affiliated with the request component words. LEADER's response will be a set of noun phrases within a single genus that contains the maximum number of request component words.

Consider the following simple example. Suppose that a user's request was reduced to the noun phrase

finite Markov chain

by LEADER. This phrase would then be further reduced to the three word components

finite
Markov
chain

Next, LEADER would look up each of these word components in the Noun Phrase-Word Profile Dictionary and the Noun Phrase Dictionary. Suppose the result was as follows:

Word Component	*Genus*	*Phrase*	*Phrase No.*
finite	3	finite chain	1
		finite Boolean lattice	2
		finite Markov chain	3
	6	bounded finite space	6
		finite set	7
Markov	3	finite Markov chain	3
		ergodic Markov chain	4
		Markov Processes	5
chain	3	finite chain	1
		finite Markov chain	3
		ergodic Markov chain	4
	11	chemical chain	8
		chain reaction	9
		finite molecular chain	10

Merging the phrases by genus and ranking them by the number of word components each phrase affiliated phrase contained, and then by frequency of occurrence of the affiliated phrases, the result would be as follows:

Genus	*Phrase*	*Phrase No.*	*No. of words*	*Frequency*
	finite Markov chains	3	3	3
	finite chains	1	2	2
3	ergodic Markov chains	4	2	2
	finite Boolean lattice	2	1	1
	Markov Processes	5	1	1

Genus	*Phrase*	*Phrase No.*	*No. of words*	*Frequency*
6	bounded finite space	6	1	1
	finite set	7	1	1

Genus	*Phrase*	*Phrase No.*	*No. of words*	*Freqrency*
	finite molecular chain	10	2	1
11	chemical chain	8	1	1
	chain reaction	9	1	1

In this case, the phrases in genus 3 would be presented to the user as a response. It would then be up to the user to select or reject this output based on what it is that he is looking for. (If it were the case that the user rejected all of these phrases, LEADER would present him with the phrases from the next highest ranked genus, 11.) Let us assume that the user is satisfied with the phrases in genus 3. He may at this point choose to see any documents associated with these phrases (or any combination of them), or he may continue his negotiation to further clarify his request. Suppose he selects the phrases

finite Markov chain	3
ergodic Markov processes	4

to continue with. LEADER will now proceed to obtain all phrases in genus 3 that are affiliated with the two selected phrases. This information is obtained from the Phrase Affiliation File and the Noun Phrase Dictionary. The two sets of affiliated phrases are then merged and ranked by affiliation value. The results may be as follows:

Phrase No.	*Phrase*	*Affiliated Phrases*	*Value*
3	finite Markov chain	finite Markov chain	10
		unique probability vector	5
		finite state automata	4
		vector algebra	3
4	ergodic Markov chain	ergodic Markov chain	9
		unique probability vector	7
		finite state automata	4
		context-free grammars	4

The merged and ranked response would be

Affiliated Phrase	*Affiliation Value Sum*
unique probability vector	12
finite Markov chain	10
ergodic Markov chain	9
finite state automata	8
context free grammars	4
vector algebra	3

Again, the user may decide to look at various document sets associated with any combinations of these phrases he may choose. Or he may again continue with these phrases to further define his request. The user also has the option of returning to any preceding step in his request negotiation at any time, or he may choose an entirely new search direction. The important point is that there is a continuous dialogue between the user and the LEADER system allowing the user to become familiar with LEADER'S file organization, and manipulating it as he wishes.

When the user decides he has reached a point at which he would like to see some documents, LEADER provides him with a very flexible documental unit display feature. He may choose to see: author/title information

citations
full text

In the case of full text display, the user may view the complete document, if he desires, in a continuous display, or he may choose to see all those sentences in a given document which contain one or more of the phrases he has selected earlier. Finally, he may choose to see sentences from each document presented to him containing associated phrases. Thus the user has complete browsability free of language and hardware use restrictions.

ACKNOWLEDGMENT

Grateful acknowledgment is made to the National Science Foundation for support under Grant No. GN-668 of the work on which this paper is based.

REFERENCES

1 (i) D J HILLMAN
Characterization and connectivity
Document Retrieval Relevance and the Methodology of Evaluation National Science Foundation
Grant No GN-451 Report No 1 May 24 1966

(ii) W R HILTON D J HILLMAN
The structure of LECOM
Ibid June 29 1966

(iii) D M REED D J HILLMAN
Microcategorization for text-processing
Ibid July 7 1966

(iv) D M REED D J HILLMAN
Canonical decomposition
Ibid August 12 1966

2 (i) *Problems, systems and methods*
Study of Theories and Models of Information Storage and Retrieval Grant G24070
The National Science Foundation August 3 1962

(ii) *The Boolean algebra model*
Ibid

(iii) *A positive model for systems of special classification*
Ibid August 29 1962

(iv) *New foundations for retrieval theories*
Ibid August 12 1963

(v) *Positive models of retrieval systems as species of logical algebras*
Ibid August 23 1963

(vi) *Retrieval systems for non-static document collections*
Ibid September 26 1963

(vii) *Graphs and algorithms for term-relations*
Study of Theories and Models of Information Storage and Retrieval
The National Science Foundation Grant No GN-283
July 30 1964

(viii) *The structure of document relations*
Ibid August 25 1964

(ix) *Topology and document retrieval operations*
Ibid July 1 1965

(x) *The formal basis of relevance judgments*
Mathematical Theories of Relevance with Respect to the Problems of Indexing National Science Foundation Grant No GN-177 July 9 1964

(xi) *An algorithm for document characterization*
Ibid March 12 1965

3 *Cf.* Reference 2.

System characteristics of Intrex

by J. F. REINTJES

Massachusetts Institute of Technology
Cambridge, Massachusetts

The salient features of Intrex have been reported in prior literature.[1] They are updated here and are included as a preamble to the three companion papers being presented at this conference on several technical features of Intrex.

Intrex is an experimental, pilot-model, machine-oriented library system. As illustrated in Figure 1, the system includes a computer-stored catalog of ten-thousand journal articles in selected fields of materials science and engineering, and the full text of the ten-thousand articles stored on microfiche. The catalog is contained in a general-purpose time-shared computer and is accessed through specially designed alphanumeric consoles, one of which has been implemented to date. The consoles are connected to the central computer through a buffer/controller. The full-text microfiche collection is accessed through the alphanumeric console and the access facility is designed to provide guaranteed, rapid access to any documen in the collection at locations which are remote to th store. Full text may be viewed, page-by-page, at the user's station by means of a separate storage-tube display, or a permanent copy may be obtained either on 35-mm film or as an 8½ × 11 inch print. The elapsed time from the ordering of a 35-mm film copy to the availability of it at the film station is approximately 90 seconds. The first page of an article appears on the storage-tube display within 7 seconds after an order is placed and each succeeding page can be obtained within 3 seconds.

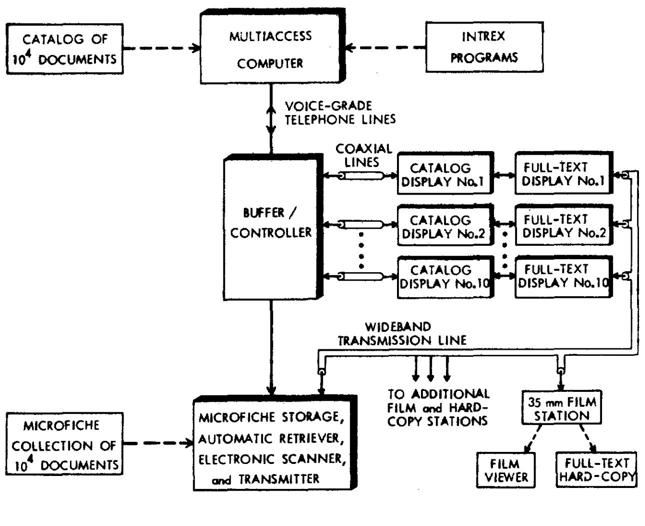


Figure 1—System diagram of Intrex

Our objective in Intrex is to use this experimental library as a means for gaining insights into the design characteristics of large-scale, operational systems of a similar kind. We are in the process of evaluating the merits of the system by making it available to a selected community of users who have a bona fide need for the information contained in the system and to librarians who may wish to use it for reference purposes. It is our intention to alter the characteristics of the system as we learn about its strengths and weaknesses from our user community.

It should be noted that the system illustrated in Figure 1 brings the library to the user; it circumvents his need to go to the library for the information he is seeking. To be acceptable as a working system, however, it must engender satisfaction from the viewpoints of completeness and relevance of the information retrieved; it must be easy to engage; and it must provide quick, reliable service at costs that are realistic.

Intrex is examining several issues with respect to in-depth cataloging and the extent to which in-depth cataloging is needed when guaranteed, rapid access to full text is, and is not, readily available at the user's station. As many as 52 different items of information are being entered for each journal article cataloged. These items are described together with access procedures, in a companion paper by R. S. Marcus et al.[2] Through monitoring of the frequency of requests for each item we shall be able to draw conclusions on the relative value of each item.

The possibility exists that easy access to full text may alter the behavior patterns of those using the catalog. Since some of our catolog information is frequently contained on the first page of the document itself (author, author's affiliation, journal name, volume number, page, abstract, and so forth), users have the option of obtaining this information either by retrieving it from the catalog or from the document itself. Our purpose is to investigate the factors which govern user behavior when seeking information of this kind.

For storage of full text we have chosen image storage on microfiche. Computer storage of full text was discarded because of the huge amount of storage required, particularly when gray-level information must be preserved. The use of film ensures preservation of pictorial information, and microfiche is well-suited for journal-article-type literature. We are using COSATI standard microfiche with a reduction ratio of approximately 18 to 1 and sixty frames per microfiche. Since our documents are derived from the published literature, we store text in image form rather than digitally because of the ease with which page text can be converted to a microfilm image.

The full-text storage and scanning unit, called the central station, is time-shared by several receiving stations. In order to minimize the time each microfiche is out-of-storage, each frame on a microfiche is scanned only once by means of a flying-spot scanner. Video signals are transmitted as analog information and to each frame of transmission is added a receiving-station address in digitally encoded form. Single-frame transmission requires storage at the receiving stations; this storage is photographic in the case of the film station and electronic in the case of the storage-tube receiving station.

Our investigations show that at least 2,000 scan lines are required to reproduce the stored images with acceptable legibility.[3] Variations in type sizes and quality of printing among documents dictate this scan-line requirement; in fact, as many as 3,000 lines may be needed where the quality of print is marginal and small type size is present, as in the superscripts and subscripts of mathematical equations. This high-resolution requirement would require an extraordinarily wide-band transmission system if a television type repetition scan were employed. With one-shot scan, as used here, a trade-off can be made between bandwidth and scan time per frame. Approximately a 1-MHz channel with a scan time of 2 seconds has been found compatible with other elements of the text-access system. Details of the text-access and transmission system are contained in the paper being presented at this conference by D. Knudson and S. Teicher.[4]

The alphanumeric-console system through which the catalog is accessed has been developed as an experimental vehicle to determine special attributes which should be included for user convenience in library applications and to investigate techniques which might be employed to minimize cost of such consoles. The salient features of the catalog console system are these: typewriter keyboard input; refreshed CRT-display output; and a buffer/controller, consisting of a 128-track drum storage device and a Varian Data Machines 6201 computer, interposed between the central computer and the display consoles. The buffer/controller is capable of serving a maximum of ten display terminals. In addition to the keyboard, a set of programmed and programmable buttons is being provided in an effort to determine the usefulness of this approach as a user aid.[5]

In the original system design, catalog information and full text are displayed on separate CRT's. This configuration is obviously inconvenient and costly; a single display is a goal of our project. Such a display is described in the paper being presented at this conference by J. K. Roberge and D. R. Haring.[6]

System utilization

A critical item with respect to utilization of a machine-stored library is the amount of preparation needed to engage the system and to make full use of its power. The bona fide user is interested only in satisfying his need for information as completely and quickly as possible, and with a minimum amount of preliminary effort. Nevertheless certain procedural matters must be mastered even before intelligible responses can be derived from the machine. These include an understanding of the options the user has for making searches and typing procedures for executing these searches, as well as an ability to type and to invoke proper procedures for correcting typing errors. Since many library users are occasional or intermittent users, they may always be in the category of "new" users in the sense that their retentivity of basic operational rules from system engagement to system engagement may be minimal. User aids thus become a crucial item.

Thus far, Intrex has experimented with several types of user aids. A User Guide which describes in detail the various features of the system and how to use them is available both *off-line* in hard-copy form, and *on-line* as a computer printout. In addition, the

Guide is available in summary form in a separate booklet entitled *How to Get Started.* Wall charts describing system operation are also posted directly before the user above his console. Still to be prepared and tested are simple instructions on cards.

The results of our experiments to date indicate that the off-line Guide is the version referred to most frequently; apparently, the time required to print out the various sections of the on-line version is considered to be wasteful. Nevertheless dissatisfaction with the off-line Guide has been expressed. Dissatisfaction seems to result from the large quantity of material it contains, the time required to assimilate the material, and the fact that the phraseology includes a certain amount of technical jargon which is not understood. A further observation is that a summary version of the Guide is helpful, but it must be supplemented by the full-scale version. Finally, response to wall charts as user aids has been disappointing thus far. Further experimentation with their content is needed in order to determine their value as a user aid, if indeed they have value.

To date experiments have been conducted only with a typewriter console as an input-output device. A new dimension to user-aids is added when a graphical terminal becomes available. Since its writing rate is an order of magnitude faster than that of the typewriter, current disinterest in the on-line version of the instructional guide may disappear. Furthermore, since the Intrex graphical console is being designed with a set of programmable switches, these may offer a wholly new approach to on-line instruction when these switches are programmed as user aids.

ACKNOWLEDGMENT

The research reported in this paper was supported through grants from the Council on Library Resources, Inc., the National Science Foundation, and the Carnegie Corporation.

REFERENCES

1 C F J OVERHAGE J F REINTJES
Information transfer experiments at M I T
Proc IFIP Congress 1968 Applications 2 Booklet G 18–22

2 R S MARCUS P KUGEL R L KUSIK
An experimental computer-stored augmented catalog of professional literature
Proc S J C C 1969

3 D R KNUDSON S N TEICHER J F REINTJES U F GRONEMANN
Experimental evaluation of the resolution capabilities of image transmission systems
Information Display September/October 1968 31–43

4 D R KNUDSON S N TEICHER
Remote test access in a computerized library information retrieval system
Proc S J C C 1969

5 D R HARING
A display console for an experimental computer-based augmented library catalog
1968 A C M National Conference and Exposition
Las Vegas Nevada August 27–29 1968 (To appear in Conference Proceedings)

6 J K ROBERGE D R HARING
A combined display for computer-generated data and scanned photographic images
Proc S J C C 1969

An experimental computer-stored, augmented catalog of professional literature

by RICHARD S. MARCUS, PETER KUGEL and ROBERT L. KUSIK

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

This paper reports on progress in the development and application of computer programs for storage and retrieval operations associated with the Project Intrex augmented catalog experiments. A general review of Project Intrex and details of other Intrex developments are given in companion papers.[1-3]

The environment for which these programs are currently being developed includes the following features:

computer system: time-sharing computer utility with satellite computer links emphasizing man-machine interaction via typewriter and display consoles.

data base: an in-depth catalog to a moderate-sized (10,000 documents), but growing, data base featuring free-vocabulary indexing and a variety of document types.

experiments: experiments with researchers using the system to satisfy real information needs to determine the relative values of system features

While this mix of features is, perhaps, unique, there have been, of course, many other efforts having several of these features as components. Our own work is, in many ways, a blend of various techniques found in one or more of these efforts. Among those systems having the greatest influence on the design of our system are systems implemented at SDC,[4] NASA (Bunker-Ramo),[5] University of Pennsylvania,[6] Harvard-Cornell,[7] MIT Project TIP,[8] Stanford,[9] and Bolt, Beranek and Newman.[10]

Because of the complicated nature of our catalog and an emphasis on untutored users, some techniques like instructional aids and a stemming algorithm, have been developed to a greater degree than they have in the cited efforts. On the other hand, because of a desire to get experiments under way as soon as possible some techniques, like an automated thesaurus and sophisticated matching criteria, have been deferred until experience has been gained with initial experiments.

Creation of catalog data

Data base and literature selection

The literature base for the augmented catalog experiments is drawn from the broad subject area of materials science and engineering, thus providing an interdisciplinary subject area within the scope of the M.I.T. Engineering Library which is to serve as the experimental laboratory for Intrex. Because the current literature for this entire field greatly exceeds the 10,000 document initial size of the first experimental catalog, only literature in selected areas of materials science and engineering is cataloged. These selected areas reflect the research interests of particular groups at M.I.T., thus assuring Intrex of a specific population of experimental users. Five research groups have been selected, two in physics and three in metallurgy.

The documents for this data base have been selected primarily (so far) from the journal and conference proceedings literature after January 1, 1967. (Eventually, we intend to include significant numbers of other literature forms; e.g., books, reports, theses, memoranda, etc.) Each research group chooses the journals of interest to it. Then members of each group select articles relevant to their group from journal-issue tables of contents.

The catalog and its fields

The augmented nature of the catalog is indicated by the range of the 50 fields shown in Table I. (See Benenfeld[11] for a more complete description of the nature and preparation of the catalog.) Of course, only a fraction of the fields are applicable for the catalog record of a particular document. A typical catalog record, as typed for input to the computer, is shown in Figure 1. It may be noted that some fields contain encoded information; e.g., in field 36 "e" stands for "English."

The most important field in terms of retrieval is, perhaps, the subject index terms field. Our current procedures call for generating index terms based upon the text of a document. In general, terms are combinations of phrases. Each term is structured to provide sufficient context so that the term may be understood in its own right. Further, each term is given a "range" number to reflect that proportion of a document devoted to discussing the represented concept (see Field 73 of Figure 1).

Several features of this indexing may be noted. In the first place, "free vocabulary" is used; that is, the indexer may choose any words to make up the subject terms and is not restricted to an "authority" list of terms. In practice, the indexer primarily chooses terms in the author's own words. Secondly, the meanings of, and relationships among, the terms, which in some other retrieval systems are given explicitly by a formal system of "roles" and "links," are implicitly expressed in the Intrex terms through the context given by these relatively long terms.

The rationale behind using free vocabulary and long "stand-alone" terms is one of ease of indexing and flexibility. It is evidently easier to use the author's own words to indicate the subject content than to have to re-analyze the content in terms of a fixed authority list and previously established set of roles and links. Presumably automatic indexing could be more readily developed on this basis. Similarly, there seems to be a greater flexibility in using natural language, which adapts naturally to changing conditions and conventions of meaning, rather than using authority lists, whose organization is always tending to lag behind current usage. Another feature of "stand-alone" terms is that they can be displayed to a user and, hopefully, give a more comprehensive understanding of the subject content than a list of keywords. Some experiments indicating the value of free-vocabulary indexing have recently been done by Shaw and Rothman.[12]

One disadvantage of the natural language approach is the possible reduction in retrieval effectiveness due to the diversity in ways of expressing the "same" subject. Means to circumvent this problem are discussed in the section on retrieval below. Another disadvantage of the long, stand-alone terms is their redundancy, which requires additional storage in the computer. On a purely keyword basis the redundancy is about 66 percent; that is, for each document each unique word (stem) type is used about three times in the subject terms. Determining the usefulness of this redundancy

```
//1/     4299
//2/     A24
//5/     7-C10-D3
//20/    1
//30/    1
//69/    1e(4)
//33/    illus.
//36/    e
//37/    f
//65/    b
//66/    3
//31/    dd
//47/    pp. 1453-1458. IN 8+41808
//24/    An internally reflecting optical resonator with confocal
         properties
//21/    Holshouser, D.F.
//22/    Unversity of 'Illinois', 'Urbana'. Electrical Engineering
         Dept.
//40/    'U.S.' Air Force Office of Scientific Research
          #AF-49-(638)-556 #AFOSR-62-250
//68/    Contains diagram of geometry for confocal internal reflection
//70/    Optical resonators using spherical mirrors, e.g. confocal
         systems, have been shown to have significant advantages
         over configurations using planar mirrors. In particular,
         diffraction losses can be much lower and alignment is less
         critical. However, planar systems have had an advantage
         heretofore in that coated mirrors could be replaced by
         internally reflecting prisms, thereby eliminating the
         problems associated with lossy or fragile coatings.
         Also, undesired modes are reduced since rays not parallel
         to the axis are not completely reflected. This paper
         describes the configuration for an internally reflecting
         surface which exhibits properties of a spherical mirror,
         and presents experimental results obtained with a
         semi-confocal maser using this configuration.
          [text, p. 1453]
//73/    internally reflecting optical resonator with confocal
                  properties (1);
         configuration for an internally reflecting surface which
                  exhibits properties of a spherical mirror
                  confocal system (1);
         basic properties of a confocal resonator (2);
         analytic expression for the internally reflecting surface
                  which satisfies confocal requirements (3);
         Schott barium crown glass doped with neodymium (4);
         fabrication of semi-confocal optical maser (3);


//3/     37/2,    032968,           11:15 - 11:25;
         1/1,     040168,           9:26 - 9:29;
         1/7,     040268,           11:06 - 11:08;
         11/4     040468,           11:35 - 11:50;
```

Figure 1—Sample catalog record

in aiding retrieval effectiveness is one object of our experimental investigations.

A third feature of the indexing philosophy is the extended depth of indexing. Not only are the major subjects of the document indexed but so also are subjects covered to a lesser degree or mentioned only briefly in the document. Of course, these minor subjects are given a lesser range number reflecting the smaller portion of document devoted to them. A typical journal article of five pages may be indexed by about 20 terms, each term averaging about five words. Assuming about 400 words per page, this represented an index word to text word ratio of about 0.05.

The index terms (Field 73) make up about 30 percent of the bulk of the catalog records. The abstract (Field 71) or the excerpts (Field 70)—usually only one or the other is present—make up about 40 percent of the catalog record for a given document. The other fields comprise the remaining 30 percent.

Computer system and library facilities

The Intrex Storage and Retrieval programs are presently operating in the environment of the M.I.T.-modified IBM 7094 Compatible Time-Sharing System (CTSS).[13] CTSS includes:

1. 32K (36 bit) word core for the system supervisor;
2. Another 32K core for the user working programs;
3. A high-speed drum for core images of user programs that have been "swapped out" awaiting I/O or additional service from the CPU as allocated through the service queue;
4. A low-speed drum for storage of directories to user's files;
5. Two IBM 2301 disc files (40,000,000 words each) serving as the primary storage medium for program and data files;
6. An IBM 7750 communications interface for servicing I/O needs to user consoles; and
7. Magnetic tape drives for auxiliary reading and writing of large files.

Approximately 200 typewriter consoles (including IBM Models 2741 and 1050, and Teletypewriter Model 37) and several Computer Display ARDS display consoles[14] are located on or near the campus. (Of course, consoles can be located wherever telephone lines exist.) Approximately 30 consoles can be connected to CTSS at any one time, although heavy demand by several consoles can reduce the number that can be handled practically at any one time to about 20. Under moderate loading conditions—15 to 20 consoles with moderate demand—typical response time (from the end of a user statement to the beginning of the typed response) is about five seconds. User programs typically run in time slices of up to 2, 4, or 8 seconds.

Over the time period of the work reported in this paper neither the text-access equipment[3] nor the Intrex display console[2] was available. Therefore, the bulk of the work reported concerns use of the IBM 2741 typewriter console with some use of the Computer Display ARDS display console. Of course, planning has taken this equipment into account in the retrieval programs (see below).

The M.I.T. Engineering Library is being physically reconstructed and expanded to provide an operational environment in which regular library users can experiment with the facilities of Project Intrex. Of course, it is also planned to use consoles in or near the laboratory facilities of our user groups.

Inputting and editing of catalog data

Friden 2303 Flexowriters are used to produce machine-readable punched paper tapes of the catalog records simultaneously with typed copies. A paper tape file of 10 catalog records is read into the CTSS IBM 7094 computer through a satellite PDP-7 computer in which Flexowriter codes are converted into ASCII codes. (One of the reasons for choosing ASCII for internal representation was to allow for upper and lower case alphabetic characters.) The file is stored on disc and also output on magnetic tape for printing on an IBM 1403 line printer, equipped with an extended character-set print chain. The printout is returned to the catalogers who proofread and mark errors. Correction of errors in the computer-stored working file is done by a typist at an IBM 2741 console using an online context editing program. It takes about three seconds of computer time per catalog record to perform the editing process. On the basis of our present error rate of 1.05 errors per catalog record, our error-correction cost is approximately twenty-five cents per entry. This amount represents computer time only; the typist's time (about 2.3 minutes) and the proofreader's time must also be added to determine total error-correction cost.

An analysis of the economics of replacing paper tape input with the use of online inputting showed that in the present CTSS environment online inputting would be much more expensive. Present inputting costs—both for man and machine—run about $2.50 per catalog record whereas the estimated costs for online inputting would be over $4.00 per record. The cost differential is largely due to increased computer processing time in the online mode. It is planned to redo the analysis when the Intrex Console[2] with its buffer/controller satellite computer becomes avaliable.

Computer files for retrieval

File organization (See Figure 2)

The files are organized on two levels and permit searching on three levels. The first level of file structure consists of the inverted files. An entry in the inverted files is a list of references to catalog records (documents) associated with a single primary key (title or subject word stem, or author's last name). The references contain not only document numbers but also word *specifiers* and reference *attributes*. The word specifiers establish the word position within the subject phrase, the position number of the subject phrase in the subject field and the ending that has been taken off the word to form the stem. The attributes presently include parameters such as: subject-term range number; the initials of authors' given names (note that only his last name is a primary key); and document information including whether it is a whole work or part of a larger work, whether it is a textbook or review articles, whether it has been refeered before publication and whether it is of professional level (rather than for lay consumption). A small *inverted file directory* in core memory serves to localize the position of any list in the disc-stored inverted file.

The second level of file consists of the full catalog records. These records are stored on the disc file in the order created and are located by means of a *catalog directory*, also located on the disc.

Thus, a first-level search may be made on author, title, or subject terms used as the primary key. A second-level search is then possible on the word specifiers or document attributes used as secondary keys. The third search level is a search through the catalog records themselves. Thus, the speed with which a given kind of information can be found is clearly dependent upon the distribution of the information among the various levels. The determination of the optimum distribution is therefore one of the main objectives of our experimentation.

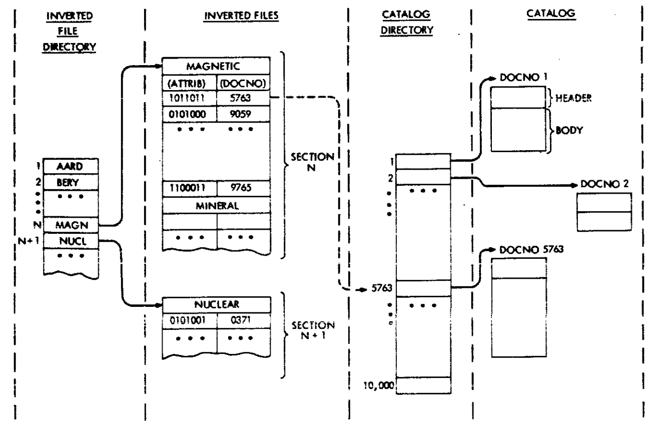


Figure 2—File organization

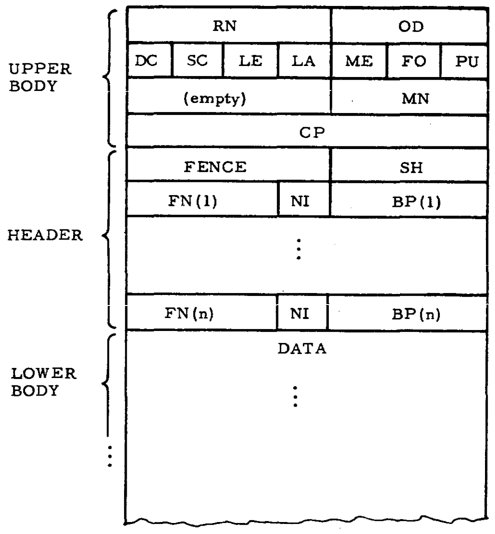
Further details of the file organization and generation are given below including observed time values for some of the critical operations. It should be pointed out that these values are a consequence of the particular software-hardware combinations we are presently using in the CTSS system (often as somewhat inefficient expedients to getting a working system going) and do not represent optimum values that are possible for magnetic-disc hardware.

Formatting and extracting

The formatting and extracting program operates on new catalog records to the system to produce three major types of output: (1) an updated main catalog-record file is produced by adding the new catalog records to the existing file; (2) index-term files are produced which contain the subject, title, and author terms and the document attributes used for updating the inverted files; and (3) a catalog-directory file is generated through which the retrieval system accesses the catalog records.

The catalog-record file is structured so that the formatting information (which indicates where records, and fields within records, begin and end) is separated from content information. The formatting information is contained in the header (Figure 3) which indicates where a given record or field begins and ends. The rest of the information is stored in the upper and lower bodies. The upper body contains the information that does not require a free format. Such information can be both compressed to save space and preformatted to simplify retrieval.

The bulk of the information in a catalog record is stored as straight text in the lower body. Characters that serve to delimit fields and record entries are removed, as are formatting characters (carriage returns and tabs). These format characters are reinserted later by the output program to fit into the various line widths of the output devices (2741 typewriter: 120 characters; ARDS display: 78 characters; Intrex console: 56 characters). The average formatted catalog record requires approximately 600 computer words for storage whereas the average preformatted record as



RN:	Record Number (Bits 0-20)	FO:	Format (Bits 25-30)
OD:	Online Date (Bits 21-35)	PU:	Purpose (Bits 31-35)
DC:	Descriptive Cataloger (Bits 0-5)	MN:	Method Number (Bits 21-35)
SC:	Subject Cataloger (Bits 6-11)	CP:	Continuation Pointer
LE:	Level of approach (Bits 12-14)	FN(i):	Field number of i-th field (address portion)
LA:	Language (Bits 15-19)	NI:	Note Indicator (tag portion)
ME:	Medium (Bits 20-24)	BP(i):	Byte pointer to bottom of field i (decrement portion)
SH:	Size of Header (in computer words)		

Figure 3—The catalog record format

input by the catalogers takes up about 700 computer words (4 characters stored per 36-bit word).

Table I—Information fields in catalog

I. CATALOG CONTROL FIELDS
1. Document Number
2. Document Selection
3. Input Control
4. On-Line Date
5. Microfiche Location

II. PHYSICAL DOCUMENT CONTROL FIELDS
10. L. C. Card Number
11. Library Location
12. Serial Holdings

III. DESCRIPTIVE CATALOGING FIELDS
20. Main Entry Pointer
21. Personal Names
22. Personal Affiliations
23. Corporate Names
24. Title
25. Coden Title
26. Edition Statement

Table I (Contd)

27. Publisher
28. Place of Publication
29. Dates of Publication
30. Medium
31. Format
32. Pagination
33. Illustrations
34. Dimensions
35. Serial Frequency
36. Language of Document
37. Language of Abstract
38. Series Statement
39. Report/Patent Numbers
40. Contract Statement
41. Supplement Referral
42. Errata
43. Thesis
44. Variants
45. Titles of Variants
46. Article Receipt Date
47. Analytical Citation
48. Abstract Services
49. Cost-Text Access
50. Commercial Cost

IV. SUBJECT CONTENT FIELDS
65. Author's Purpose
66. Level of Approach
67. Table of Contents
68. Special Features
69. Bibliography
70. Excerpts
71. Abstracts
72. Reviews
73. Subject Indexing

V. ARTICLE CITATION FIELD
80. References/Citations

VI. USER FEEDBACK FIELD
85. User Comments

The formatting and extracting program requires approximately two to three seconds to process a single catalog record.

Phrase decomposition and stemming

The subject and title terms are broken down into individual words and these words are stemmed by dropping off endings.

A two-phase stemming algorithm has been developed.[15] In the first phase, the longest possible ending from a list of about 280 endings is dropped from the word. Before an ending is dropped it must satisfy a context rule. (For example, do not drop *s* after *s*.) The second phase of the algorithm includes transformational rules to account for certain spelling anomalies in English (for example, absor*b*/absor*p*-tion; split/split-*t*-ing).

Because the transformational rules of the second phase involve various complexities in actually performing the stemming and keeping account of it in the inverted files, and because the number of additional cases it handled seemed relatively small, it was decided to try out only the first phase procedures in initial Intrex systems. A list of endings and context rules for applying them are contained in the reference document.[15]

About 100,000 subject-term words from about 1000 catalog records have been stemmed according to this algorithm. So far the results seem promising. Table II gives a statistical compilation of the number of endings found for each ending type in one run of 2,382 words stemmed.

The output from the phrase decomposition and stemming program, which takes about seven seconds per document, is a set of "shreds": one for each subject or title word and one for each full phrase not longer than a certain number of words. The maximum number of words for full phrases retained for the inverted files is currently four. This covers only about 20 percent of the terms in the catalog.

Sorting, common word culling, and merging

Sorting and merging are accomplished using a generalized sort-merge package developed by the staff of the M.I.T. Technical Information Program.[8] This package features manipulation of variable length records with a variable number of variable length keys.

The first operation involves an alphabetic sort of the shreds with the word stem or author last name as primary key and the word ending code or author's initials string as a secondary key. The secondary key is used to facilitate structuring the inverted files (see below). The distinction between upper and lower case alphabetic characters is suppressed during sorting. This operation takes about six seconds per document or about 80 msec. per index-word shred (which averages about ten computer words in length at this stage of processing).

The second operation culls out the 13 most common function words. These are listed in Table III in order of frequency. The culling operation, which takes about

Table II—Endings found in stemming 2,382 words

Ending	*Occurrences*	*Ending*	*Occurrences*
arization	9	ness	9
entations	2	ogen	20
ableness	1	wise	3
entation	3	ying	1
ability	2	age	4
ational	5	als	9
ibility	16	ant	7
ization	23	ary	5
ations	20	ate	15
encies	3	ely	1
ential	3	ene	1
istics	2	ent	30
acity	10	ial	8
aries	1	ian	8
arity	4	ics	10
ately	1	ied	4
ating	25	ier	3
ation	105	ies	16
ative	3	ine	14
ators	2	ing	83
atory	1	ion	215
ement	11	ism	4
ening	3	ity	30
ental	2	ive	5
ially	1	one	1
icity	6	ons	3
ional	10	ors	7
istic	1	ous	3
ities	4	's	24
ivity	8	al	57
able	9	ar	27
ally	1	ed	106
ance	24	en	34
ants	16	es	148
ated	12	ia	1
atic	2	ic	127
ator	7	is	12
ence	31	ly	2
ency	15	on	11
ents	30	or	11
eous	2	um	30
ible	4	a	25
ical	62	e	449
ions	40	i	23
ious	1	o	16
ized	11	s	118
less	3	y	85

Table III—Common words excluded from index lists

1.	of
2.	in
3.	the
4.	for
5.	a
6.	on
7.	to
8.	at
9.	with
10.	and
11.	as
12.	by
13.	from

one second per document, reduces the size of the files by about 20 percent.

The third operation merges the batch of sorted shreds from the latest operation with the cumulative batch of sorted shreds from previous runs. Merging takes about 0.6 seconds per document in the *total* data base.

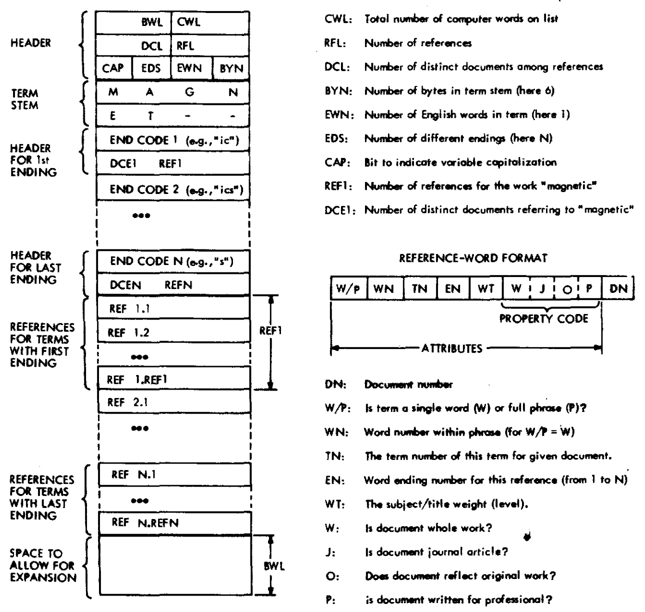
Inverted file generation and listing

The merged shreds are processed into the inverted file structure shown in Figure 4. This operation takes about 1.5 seconds per document for the combined subject/title inverted file (title words and terms are distinguished from subject words and terms by a unique range number). Generation of the author inverted files, which contain about two author names per document, takes about 50 msec. per document.

Listings can be made, for analysis purposes, of all (or sections of) these inverted files and they may contain the full references or just counts of the number of references and documents. See Figure 5 for an excerpt from a full listing. A full listing requires about three seconds of 7094 CPU time per document plus some offline 1401 time.

Retrieval procedures

An initial version of the system (see Figure 6 for a



BWL:	Number of blanks at end of list (for expansion)
CWL:	Total number of computer words on list
RFL:	Number of references
DCL:	Number of distinct documents among references
BYN:	Number of bytes in term stem (here 6)
EWN:	Number of English words in term (here 1)
EDS:	Number of different endings (here N)
CAP:	Bit to indicate variable capitalization
REF1:	Number of references for the work "magnetic"
DCE1:	Number of distinct documents referring to "magnetic"

DN:	Document number
W/P:	Is term a single word (W) or full phrase (P)?
WN:	Word number within phrase (for W/P = W)
TN:	The term number of this term for given document.
EN:	Word ending number for this reference (from 1 to N)
WT:	The subject/title weight (level).
W:	Is document whole work?
J:	Is document journal article?
O:	Does document reflect original work?
P:	is document written for professional?

Figure 4—Format for subject-term list

```
   TERM                   NO. REFS.  NO. DOCS.  NO. ENDS

cor-                        22           4          2
 core                       16           4
 cores                       6           1
 1099 21 1 1 (3) JOP     512 4 1 1 (2)  OP   512 5 7 1 (1)  OP
 512 6 7 1 (1) OP        516 6 9 1 (3)  OP   512 11 7 1 (1) OP
 516 10 1 1 (3) OP       512 3 1 1 (0)  OP   541 11 2 1 (3) OP
 512 9 3 1 (2) OP        512 1 4 1 (1)  OP   512 10 6 1 (3) OP
 516 7 1 1 (3) OP        512 0 5 1 (5)  OP   512 2 4 1 (4)  OP
 516 8 1 1 (3) OP        516 3 9 2 (1)  OP   516 10 11 2 (3) OP
 516 7 10 2 (3) OP       516 9 11 2 (3)  OP 516 8 10 2 (3)  OP
 516 4 2 2 (4) OP

cor-e steel                  2           1          1
 F512 3 2 1 (0) OP      F512 4 2 1 (2)  OP

correct-                     5           4          2
 correction                  2           2
 corrections                 3           2
 3553 2 3 1 (1)  JOP     3437 8 3 1 (3)  OP 3124  6 1 2 (3)  OP
 3634 34 5 2 (3)  J P    3634 33 5 2 (3)  J P

correl-                     21           7          3
 correlated                  1           1
 correlation                17           4
 correlations                3           3
 1030 0 3 1 (5)  JOP     •••
```

Figure 5—Inverted file listing (excerpt)

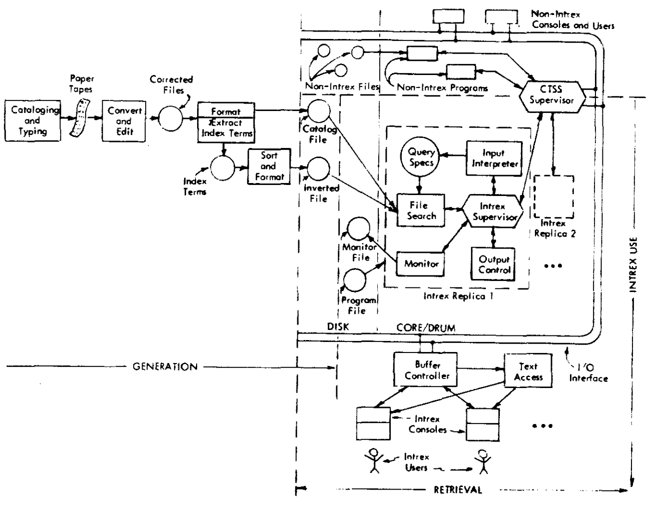


Figure 6—Intrex storage and retrieval system

diagram of the storage and retrieval programs) for the interactive interrogation of the catalog from remote consoles has been implemented. The system, termed the Prototype System, has been used, in conjunction with a data base including about 1000 documents, to begin experiments with users as described in a later section.

Description of prototype system

The Prototype System permits the user to search the data base for documents by specifying subjects, authors and/or titles. The user may then make a selection among the documents retrieved by requesting that

```
1S   TST7X5:  USERS = 16,   MAX = 47.
     READY.

1U   login m5806 marcus
2S   W 1315.7
     Password
2U
3S    STANDBY LINE HAS BEEN ASSIGNED
      M5806   160 LOGGED IN  08/19/68   1315.9 FROM 800277
      LAST LOGOUT WAS  08/15/68  944.4 FROM 800277
      HOME FILE DIRECTORY IS M5806 CMFL01

     DUE TO HARDWARE DIFFICULTIES, CTSS OPERATION MAY BE
     IRREGULAR.

       CTSS BEING USED IS TST7X5
     R 6.166+1.016

3U   resume intrex
4S   W 1316.8
     Greetings  This is Intrex 1a.  Please sign in by typing your name and
     address as in the following example:

     smith, r j/mit 13-5251

     Note that your sign-in statement should end with a carriage return.
     READY
4U   marcus, r s/mit 35-406
5S   If you already know how to use Intrex, you may go ahead and type
     in commands. (Remember, each command ends in a carriage return.)
     Otherwise, for information on how to make simple queries of the
     catalog, type

     info 2

     or, to see the Table of Contents (Part 1) of Intrex-1 Guide which will
     direct you to other parts of the Guide explaining how to make more detailed
     queries, type

     info 1
     READY
5U   info 2
6S   Part 2 of Intrex 1A Guide:  Simple Queries

             To find documents in the system specify your query by subject, author,
     and/or title terms, as shown in the 3 examples below:

     subject ferroelectric transitions

     author Hess, G. B./subject helium

     title sulfurization/author Swisher

             In order to specify additional restrictions (e.g., where author comes
     from, journal, word variations not to use, etc.), see Part 9 of the Guide
     (or type info 9). For other than standard output (document numbers, titles,
     and authors) see Part 8. For general Intrex command format and abbreviations
     see Parts 6.1 and 6.2.
     To see Table of Contents for Intrex 1a Guide and how to use
     the Guide on line, typ

     info 1

     Otherwise, you may make simple queries or use any other command.
     READY
6U   subject solid phase transitions
7S   A search on your query SUBJECT solid phas-e transit-ions found 7 documents.
     To output the catalog fields DOCUMENT NUMBER, TITLE, AUTHOR on those
     documents type

     go

     This output will take about 15 seconds per document. You may terminate
     this output at any time by hitting the ATTN key ONCE. Otherwise, you may
     change your output request. For information see Part 8 of Guide or type

     info 8

     or change your field restriction (see Part 9.5) or make another request of
     Intrex (see Part 1)
     READY
7U   go
8S   1. DOCUMENT NUMBER 2851

     (21) AUTHOR
     Hoshino, Sadao;
     Shimaoka, Kohji (JA);
     Niimura, Nobuo (JA)
```

```
     (24) TITLE
     Ferroelectricity in solid hydrogen halides

     2. DOCUMENT NUMBER 3430

     (21) AUTHOR
     Sihvonen, Y. T.

     (24) TITLE
     Photoluminescence, photocurrent, and phase-transition correlations

     3. DOCUMENT NUMBER 3174

               . . .

     7. DOCUMENT NUMBER 1690

     (21) AUTHOR
     Willens, R. H.;
     Buehler, E. (JA)
     Matthias, B. T. (JA)

     (24) TITLE
     Superconductivity of the transition-metal carbides

             Output completed. Total of 7 documents found. You may now see
     additional output on these documents by making a new 'output' request (for
     information on how to do this, see Part 8 of the guide or type info 8).
     You may also select a portion of these documents by making a new 'infield'
     request (see Part 9.5).  Otherwise, you may make a new search (see Part
     2) or make other requests (see Part 1).


     READY
8U   output affiliation matchsub relevance/go
9S   1. DOCUMENT NUMBER 2851;  RELEVANCE 3/3

     (22) AFFILIATION
     University of > Tokyo<.  Institute for Solid State Physics;
     University of > Tokyo<.  Institute for Solid State Physics;
     University of > Tokyo<.  Institute for Solid State Physics

     (74) MATCHSUB

     phase transition at low temperature in solid hydrogen halides (0).
     2. DOCUMENT NUMBER 3430;  RELEVANCE 2/3

     (22) AFFILIATION
      >Texas<  Instruments,  >Dallas<

     (74) MATCHSUB

     (TITLE)
     3. DOCUMENT NUMBER 3174;  RELEVANCE 2/3

               . . .

     6. DOCUMENT NUMBER 1715;  RELEVANCE 2/3, 2/3

     (22) AFFILIATION
     Sandia Laboratory,  >Albuquerque<,  >N. M.<

     (74) MATCHSUB

     second-order phase transition in perovskites (3);
     first-order phase transition (0);
     7. DOCUMENT NUMBER 1690;  RELEVANCE 2/3, 2/3

             . . .

     READY
9U   infield affiliation harvard/o 71/go
10S  1. DOCUMENT NUMBER 3174

     (71) ABSTRACT
     The high-temperature series expansion of the zero-field magnetic susceptibility
     *chi*/*chi**sub Curie* = 1 + *SIGMA**sub 1 = 1*sup *infinity**a*subl
     is related to the diagrammatic representation of the corresponding
     expansion of the zero-field static spin correlation function INT. 1
     Does the criREADY
10U  quit
11S  Thank you for using Intrex.
     R 109.583+20.800
```

Figure 7—Sample demonstration system dialog

their catalog records contain specified information in certain fields. Finally, he may request that the information contained in any or all of the catalog fields be printed out.

In order to illustrate the nature of the Prototype System more concretely, a sample user-system dialog is given in Figure 7. The dialog has been retyped in shortened line-width form from the typewritten copy prepared on an IBM 2741 teletypewriter console attached to the time-sharing system (CTSS). For illustrative purposes each user statement is flagged by a number and the letter *U* in the left margin. Similarly, system messages are flagged by numbers and the letter *S*.

In his first two statements the user has logged-in to the time-sharing system (CTSS) which hosts the Prototype System as well as many other computer programs. Note that the user's second statement, his password to CTSS, is not printed because the 2741 is set to the nonprinting mode by CTSS to protect the security of the password. This log-in procedure will be unnecessary for individual users situated at consoles dedicated to Intrex use.

The third user statement "resumes" the "Intrex" system (at this point the user could have called for any other program currently resident in CTSS) and in the fourth statement the user "signs in" to Intrex by typing his name and address. The "sign-in" statement, in conjunction with the monitoring procedures (see below), provides us, as system analysts, with a means for keeping track of system use. It also serves to introduce the user to certain system procedures. For example, the user is apprised of the fact that his statement must be terminated with a carriage return. (Note that a more natural *statement-terminator* button or switch will be possible with the Intrex display console.) While, at present, the sign-in statement merely serves to aid in monitoring system use, we anticipate future system developments whereby some history of past users is kept so that, when someone signs in, the system can take account of his previous experience to help direct him.

Intrex response to the user's sign-in statement, message 5S, is illustrative of several features of the prompting and instructional techniques employed by the system. In the first place, the user is told of the various alternative actions that he may take at any given time. Secondly, the specific form of the statement he should type to invoke one of these actions is explicitly stated, where possible. Thirdly where it is not possible to explain the alternatives completely, the user is referred to a Guide for further details.

```
Part 1 of Intrex 1A Guide:  Table of Contents

     To have a part of the Guide printed out on line use the
"Info" command.  For example,  for  information  on  making
simple queries (i.e., to print out part 2), type

Info 2

PART      CONTENTS
1         Table of Contents
2         Simple Queries
3         General Remarks - How to Get Guide (printed copy)
4         Log-in to CTSS and Call Intrex
5         Typing Errors - How to Correct
6.1       Commands, Modes (LONG, SHORT), Time Checks
6.2       Command Names and Abbreviations
7         Preliminary Output
8         Final Output
9         Generalized Queries
10        Scanning Index Terms
11        Interrupting System Messages
12.1      Text Access
12.2      Library Services
13        User Comments and Questions
14        Documents in the Collection
15        The Catalog and Its Fields
16        Sample Catalog Record
17        Exit from the System

     This online guide was last revised on 7/24/68.



INTREX 1A as of 24 JUL 68

Part 2 of Intrex 1A Guide:  Simple Queries

     To find documents in the system specify your  query  by
subject, author, and/or title  terms,  as  shown  in  the  3
examples below:

subject ferroelectric transitions

author Hess, G.B./subject helium

title sulfurization/author Swisher

     In order  to  specify  additional  restrictions  (e.g.,
where author comes from, journal,  word  variations  not  to
use, etc.), see Part 9 of the Guide (or type Info 9).    For
other than standard output (document  numbers,  titles,  and
authors) see Part 8.  For general Intrex command format  and
abbreviations see Parts 6.1 and 6.2.
```

Figure 8—Sections from Intrex guide

The Guide is available both in hard-copy form and online. Selected pages of the Guide are shown in Figure 8. The user may request that a section of the Guide be printed online by using the INFO command (see user statement 5U and system response 6S). The Guide also attempts to use the techniques of presentation of alternatives, example, and reference to more detailed information. The sections of the Guide are sized for convenient printing and viewing online.

The user's sixth statement initiates a search in the inverted files for documents on a given subject. Searching may also be done on title or author terms or combinations of subject, title and/or author terms. It may be noted that the form of the user's statements is a compromise between the precise, but esoteric and complicated, form of many programming languages and the familiar, but ambiguous (and, therefore, difficult to interpret automatically), form of natural English. Command and argument names are simple and mnemonic.

Format is kept simple with only three basic delimiters required: spaces to separate arguments from each other and from command names, slashes to separate commands, and a carriage return to terminate the statement.

In response to the user's search request, Intrex replies with a message (7S) illustrative of several other features of system dialog. In the first place, the system plays back its understanding of the user's statement. Also the system indicates, by hyphenating word endings, how it has stemmed the words in the user's search specification. This is important because Intrex matches these word stems to word stems in the inverted file. As a further indication of the progress of the retrieval process, the number of matching documents found in the inverted files is printed. Since the user made no special output request in statement 6U, Intrex reports the estimated time to output the standard catalog fields. This system message, then, gives feedback which may interact with the user's original intentions and expectations and allow him to redirect his search.

The points at which the system reports to the user have been chosen in light of the operating characteristics of the host CTSS time-sharing system. The intention is to report back soon enough so that the user experiences quick response but not to report so often that the user is forced into unnecessary additional responses of his own. The incorporation of the buffer-controller computer for the Intrex display consoles will improve the operating characteristics of the time-sharing environment and may allow more frequent feedback with less cost at the central-computer level.

In our sample dialog the user takes the first alternative (statement 7U) and the system responds with the standard output (message 8S) for the matching documents. The ellipses (...) in the figure indicate where portions of the system response have been deleted to reduce figure length.

At this point in the dialog, let us assume that the user already knows how to make an output request or that he refers to his hard-copy version of the Guide. In any case, the user's eighth statement requests additional output information. Note that by appending the GO command to the OUTPUT command, the user signifies he is sufficiently sure of his statement not to want Intrex to respond with its interpretation and timing estimate but rather to print the requested output immediately.

The system then responds (message 9S) as directed. Note the special output information giving those subject terms that matched (MATCHSUB) and the estimated relevance of these terms. The relevance of a subject term to a user query is currently estimated simply by the ratio of the number of words in the query which match words in the term to the total number of words in the query.

In statement 9U the user is selecting, by means of an INFIELD command, a subset of the seven documents which his original search found. This command enables the user to request only those documents in which a specific character string (here, "harvard") appears in a particular catalog field (here, "affiliation"). Note that, at the same time, the user is changing his output request and using abbreviation "o" for the command name "output" and "71" for the field name "abstract".

In the system's response (message 10S) to the above request, the user has availed himself of the interrupt capability and halted the output at the point indicated by the letters "INT 1." The system then responds with the READY message indicating the user may go ahead with other requests. The interrupt capability is a general facility allowing the user to cut short system messages. After the user has become familiar with the system he can reduce the verbosity of system messages by entering the SHORT mode. He may do this at any time during the dialog or even upon resuming the system as shown in Figure 9.

```
resume intrex short
W 1355.1
Please sign in.
R
marcus r s/mit 35-406
R
s solid phase transitions/in affiliation harvard/o 22 abstract/go
1. D 3174

(22) Lyman Laboratory of Physics Harvard University, >Cambridge<,
 >Mass.<; / Lincoln Laboratory M. I. T., >Lexington< , >Mass.<

(71) The high-temperature series expansion of the zero-field magnetic
susceptibility, *chi*/*chi**sub Curie* = 1 + *SIGMA**sub 1 = 1*sup
*infinity**a*sub 1*(J/kT)*sup 1*, is related to the diagrammatic
representation of the corresponding hh-tINT. 1
are then
R
s transint## tions/a hoshino/o 21 74 75
S: transit-ions / A: hoshino found: 1 doc o: 21, rel, msub  15
secs/doc.
R
o go
Sorry, I can't understand you.
R
go
1. D 2851

(21) Hoshino, Sadao;
Shimaoka, Kohji (JA);
Niimura, Nobuo (J A)

(24) Ferroelectricity in solid hydrogen halides

   1  docs found
R

quit
Thank you for using Intrex.
R 36.050+10.433
```

Figure 9—Sample dialog in SHORT mode

Monitoring system use

Embedded in the retrieval system is a monitoring system which records, on a disc file for later analysis, all user commands and system responses as well as certain timing information. In addition, a shared console remote from the user console may be employed to monitor experiments in real time.

With the COMMENT command a user may input a comment about the system, the cataloging, or the documents in the collection. These comments are recorded by the monitoring system. Comments about the catalog may result in modifications to the catalog at some subsequent update whereas comments about the documents may get entered into Field 85 (see Table I) of the pertinent catalog records.

User experiments

The users

When the size of the data base reached about 1000 documents, experiments utilizing the Prototype Retrieval System were begun with users having a real need for information. The first user was a second-year graduate student in physics (from the first of the research groups mentioned in an earlier section) who was starting a project to measure the magnetic susceptibility of europium sulfide near the critical point. He had already compiled a bibliography on this subject through conventional library techniques but was still seeking information on the light absorption properties of EuS to properly set up his experimental equipment. Six additional users were taken from the ranks of the Intrex catalogers by giving them a description of the student's problem and asking them to serve as reference librarians using the Intrex system.

Experimental environment

Users were seated at a 2741 console with no personal instruction. They had previously been given a hard-copy version of the Guide (at least a day in advance) to which they could refer during the retrieval session. Other user aids (besides the system dialog as exemplified in Section 6), included messages pasted on the console (e.g., "Don't forget the carriage return"); wall charts (to remind the user how to perform common functions); the NASA Thesaurus[16] (to suggest semantically related words for user search requests); and the Inverted File listings (to suggest additional search words as well as show document counts for index terms). Users were given extensive debriefings (up to an hour and a half) by systems analysts after their console sessions (which lasted between about one-half and one hour).

Results

Results of these first experiments are still under analysis and, in any case, the small size of the sample user population and still modest size of the data base make it clear that these "results" can only suggest future lines of investigation and cannot provide definitive conclusions. With these caveats in mind we make the following tentative observations:

1. Users learned to operate the basic features of the system fairly easily and found all or most of the relevant documents rather quickly.
2. The users, in the hour or so of their acquaintance with the system, mastered few of the sophisticated features of the system nor did they really understand the nature of the matching algorithm.
3. Users without previous computer experience tended to be awed by the computer which inhibited their trying, and learning, system features.
4. The difficulties listed under (2) and (3), while not hindering user retrieval seriously for the sample problem, could adversely affect results on other problems and could degrade our efforts to determine the relative merits of various augmented-catalog features.
5. One possible solution to some of these problems is to advance the user's understanding in stages by starting with simplified guides or with personalized instruction.
6. Users dislike and are confused by command mnemonics that are not single English words (e.g., MATCHSUB, INFIELD).

Future plans

Enlarging the data base, improving system efficiency, improving user aids (see previous section), and expanding user experiments are high on the list of planned projects. Incorporation of the Intrex console and text access systems into the retrieval system is also an immediate prospect. The first full version incorporating all Intrex subsystems will find the Intrex console operating in a "transparent" mode so that this console will look like a standard CTSS console and present retrieval programs can be used essentially unchanged. As experience is gained with this simple configuration an attempt will be made to shift some CTSS operations to the satellite computer associated with the Intrex consoles.

As mentioned in the Introduction, a number of system features have been deferred in order to get the Prototype System into operation as soon as possible. Some of these features are listed below. The schedule of their incorporation will undoubtedly be determined to some extent by our experimental findings.

One major area under study is the question of matching algorithms and relevance. As indicated above, matching is done on a modified "anding" of all query terms. One would like the user to have the ability to specify any combination of "ands", "ors", and "nots" among query terms, to control the relative emphasis on words within the search specifications, to make online modifications to the relevance and matching criteria (by term ranges, for example), and to selectively override the stemming and phrase decomposition algorithm.

Other improvements being planned include: searching restrictions on document properties or subject term ranges; decoding of catalog fields (for example, "English" for "e") on output; more general INFIELD specifications (for example, ranges on dates); online display of inverted file terms and frequency counts; naming lists of documents or commands for later reference; and an overlay procedure for reading in sections of the retrieval program from disc storage as the overall size of the retrieval system expands beyond core memory size.

Related work

Two further capabilities that we hope to incorporate into future augmented-catalog experiments have been studied by two students working toward M.S. degrees. Mr. Richard Domercq[17] has studied the automatic derivation of synonomy and hierarchical relationships among the subject terms on the basis of co-occurrence. Mr. William Kampe[18] has investigated automatic methods for deriving subject terms from the title and abstract of a document.

Throughout the work on storage and retrieval of catalog data we have been conscious of the problems that will be encountered in scaling up a computer-stored augmented catalog by two orders of magnitude. For a collection of one million documents, we estimate that the total information stored in the catalog will be of the order of 2×10^{10} bits. About 15 percent of this information will reside in the inverted files. A preliminary study of file organization, cost, and speed of response of such a catalog has been conducted by Professor A. K. Susskind.[19] He has arrived at a conceptual design that will perform inverted-file searches at an average rate of 40 per second with a storage device that costs about $250,000. On-line storage of the complete catalog entries appears prohibitively expensive with today's technology, and new mass-storage concepts must be examined. Multiple reading-head, high density, continuous motion magnetic tape devices appear promising and are being studied.

ACKNOWLEDGMENT

The research reported in this paper was supported through grants from the Council on Library Resources, Inc., the National Science Foundation and the Carnegie Corporation.

REFERENCES

1 J F REINTJES
System characteristics of Intrex
Proc S J C C 1969

2 D R HARING J K ROBERGE
A combined display for computer-generated data and scanned photographic Images
Proc S J C C 1969

3 D R KNUDSON S N TEICHER
Remote text-access in a computerized library information-retrieval system
Proc S J C C 1969

4 H P BURNAUGH
The BOLD (bibliographic on-line display) system
In: Schecter George ed. Information retrieval—a critical review Thompson Washington D C 1967 53–66

5 D J SULLIVAN D M MEISTER
Evaluation of user reactions to a prototype on-line information retrieval system [RECON]
In: Proc of the 30th Annual Meeting of the American Documentation Institute New York October 1967

6 M RUBINOFF S BERGMAN W FRANKS E R RUBINOFF
Experimental evaluation of information retrieval through a teletypewriter
C A C M Vol 2 No 9 September 1968

7 G SALTON
Automatic information organization and retrieval
McGraw Hill New York 1968

8 M M KESSLER
The "on-line" technical information system at M I T Project TIP
In: 1967 IEEE International Convention Record Institute of Electrical and Electronics Engineers New York 1967 part 10 40–43

9 E B PARKER
Stanford physics information retrieval system (SPIRES) Annual Report
Stanford Institute for Communications Research Stanford California December 1967

10 S I ALLEN G O BARNETT P A CASTLEMAN
Use of a time-shared general-purpose file-handling system in hospital research
Proc IEEE Vol 54 No 12 December 1966

11 A R BENENFELD
Generation and encoding of the Project Intrex augmented

catalog data base
Proc of the 6th Annual Clinic on Library Applications of Data Processing University of Illinois Urbana Illinois May 7 1968

12 T N SHAW H ROTHMAN
An experiment in indexing by word-choosing
Journal of Documentation Vol 24 No 3 September 1968

13 R M FANO
The MAC system: the computer-utility approach
IEEE Spectrum January 1965

14 R H STOTZ
A new display terminal
Computer Design April 1968

15 J B LOVINS
Development of a stemming alogrithm
M I T Electronic Systems Laboratory Technical Memorandum ESL-TM-353 June 1968 (To appear in MT Vol 11 No 1 March 1969)

16 *NASA thesaurus*
NASA report SP-7030 Scientific and Technical Information Division National Aeronautics and Space Administration December 1967

17 R J DOMERCQ
A machine-aided thesaurus generation system
M S Thesis Electrical Engineering Department M I T September 1967

18 W R KAMPE
Pre-indexing by machine
M S Thesis Electrical Engineering Department M I T June 1968 also Electronic System Laboratory Report ESL-R-355 July 1968

19 INTREX Staff
Project Intrex Semi-annual Activity Report PR-4 M I T September 15 1967

Remote text access in a computerized library information retrieval system

by D. R. KNUDSON and S. N. TEICHER

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

As part of the Intrex experiments with a computerized model library,[1] we are developing an experimental system for accessing the full text of the 10,000 journal articles in the Intrex data base. The essential result of the user dialog with the augmented catalog is the identification of those articles selected as relevant to the user's inquiries.[2,3,4] The next step in the library-usage process is to retrieve the text of these articles for reading. The convenience, response time, and quality of the text-access system is likely to affect the extent to which the user negotiates with the catalog before requesting full text. The Intrex system will permit experimentation with the complete machine-aided library operation. The purpose of this paper is to outline the text access problems and our approach to them.

The goals of the text-access system are to provide guaranteed, rapid access to full text at remote locations. Guaranteed accessibility implies a controlled, central store where the text always remains in the system and is available to users at all times. Rapid accessibility at remote locations implies transmission of text in electrical-signal form. Remote accessibility of text and catalog information from locations near the user's working area provides more convenient library use, and in addition is a preliminary step toward realization of a network of computer-based libraries coupled together by data communication links.[4,5]

Textual images differ from the displays commonly encountered in computer-oriented systems. A page from a typical technical journal contains from 5,000 to 10,000 character spaces and is composed from an unrestricted character set including foreign alphabets, mathematical symbols, and sub- and super-scripts, etc. Text frequently contains graphics and pictorial information with several gray levels, and in some instances, with color. Even the text layout and type font may influence the effectiveness of its information transfer.

These special characteristics of textual images restrict the types of displays that are appropriate in a text-access system. The graphics, photographs, and unrestricted character sets in text lead to raster-scanned displays rather than character- and vector-generated displays. The gray levels in photographs require a display with multi-level intensity. The large number of characters and symbols requires a high-resolution display, and much of our early efforts were aimed at establishing quantitative resolution requirements for textual images.[6] A high-resolution color display presents a real challenge to the display designer.

A wide variety of systems have been developed for storing, transmitting, and displaying textual images.[7,8] Slow-scan facsimile techniques, where the image is scanned and transmitted over common-carrier lines, is frequently used for document transmission. The Alden/Miracode system is an example which uses microfilm storage and slow-scan facsimile transmission. The disadvantage of these systems or our application is that several minutes are required to transmit each page and there is no capability for rapid scanning of documents before requesting hard-copy. Closed-circuit TV has been used in some systems for remote viewing of text. These systems generally operate at bandwidths of 20 MHz or less which limits the resolution of the displayed image. The Remington-Rand Remstar system is an example of the use of closed-circuit TV for transmitting images from microfilm to remote terminals. This system provides a zoom capability for viewing an enlarged section of the image in order to compensate for the resolution limitations. The Ampex Video File is an example of a system which uses magnetic tape for storage with the capability of providing a soft- or hard-copy output.

The storage and transmission methods adopted for the text-access system are pivotal parts of the system design and the considerations leading to microfilm storage and single-frame transmission are discussed below.

Single-frame transmission

A facsimile-like system, where each page is scanned and transmitted only once per request, has two distinct advantages over a system which continually transmits the image, as is done in closed-circuit TV. With the resolution required to display a full page of text, a TV-type refreshed image requires a video-signal bandwidth of approximately 60 MHz for a flicker-free display. Also, a separate scanner and transmission line is required for each simultaneous user. By providing image storage at the receiving terminals, single-frame transmission can be used which permits a trade-off between video-signal bandwidth and the transmission time per page. If the retrieval and transmission times are short compared to the average reading time, the scanner and transmission network can be time-shared among several on-line users. With a transmission time of two seconds, a 2,000 scan-line image requires a bandwidth of approximately 1 MHz. If the average reading time per page is one minute or greater and the total retrieving and scanning time is 2–3 seconds, reasonable service for a number of users could be provided from a single scanner and transmission network. The lower bandwidth and time-multiplexing capability are the key arguments favoring a single-frame transmission system.

Digital versus facsimile storage

Facsimile storage of text implies that the data base is stored as duplicates such as photographs of the original documents. Digital storage implies that the data base is stored in arrays of binary numbers that can be decoded by a suitable algorithm to reconstruct the original document. An advantage of the digital form is that if the text is properly encoded it may be computer processed for such purposes as automatic extraction of bibliographic information and automatic fact retrieval. For materials containing a limited character set, such as typewritten text, digital encoding can be more efficient in the use of storage space than facsimile storage because much of the page image contains little information. Efficient encoding can also reduce the bits per page that must be transmitted, thus saving on communication cost.

At present, textual material is not readily available in digitally-encoded form and the conversion costs are a significant obstacle to digitally-stored text. Further development of optical character readers or the availability of text in digital form directly from publishers could reduce the cost of preparing the digital store. However, the dificulties in the digital encoding of pictorial information, unrestricted character sets, and various layouts is likely to preclude digital storage for full text in the near future, particularly if no machine processing of the information is required.

The Intrex system utilizes digital storage for the augmented catalog (which requires computer processing) and facsimile image storage on microfilm for full text.

The COSATI microfiche format was chosen as the storage form for textual images because suitable retrieval equipment was available for this form and the reduction ratio required to store an entire page of typical journal article text within one frame of the COSATI grid is compatible with the resolution requirements of the text-access system. Each 4 × 6-inch microfiche contains a maximum of sixty frames and each frame contains an image of one page. The frames are located in five rows, and each row contains twelve frames. To facilitate automatic retrieval and scanning, extra care was taken during the microfilming process to achieve minimum tolerances in the location of the individual page images with reference to the COSATI grid.

Resolution

In the initial phase of the Intrex program, it became apparent that some experimental work was necessary to establish the resolution and the number of scan lines required in the text-access system; therefore an experimental image-transmission system was assembled. It utilized a flying-spot scanner to convert a microfilm image into a video signal which was transmitted over a coaxial cable. At the receiver, the image was reconstructed on a CRT and filmed with a 35-mm camera. This system is illustrated in Figure 1.

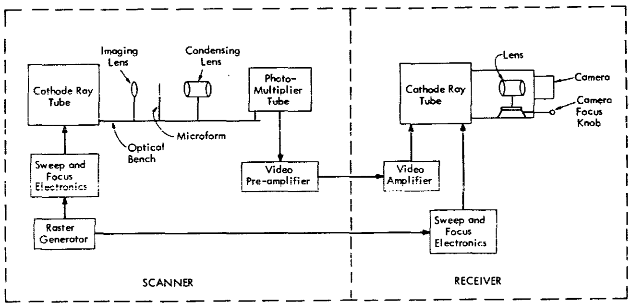


Figure 1—Experimental image-transmission system

The Modulation-Transfer Function (MTF) concept was used for quantitative resolution measures of the individual components and for relating these to the over-all system resolution. Experiments were conducted in which selected text was scanned and transmitted through the system under various MTF conditions and with different numbers of scan lines.[6] Evaluations of the transmitted images demonstrated that a minimum limiting resolution of 1000 cycles/page and at least 2000 scan lines/page are required for reproducing text from typical technical journals with acceptable image quality. Resolutions or raster scans below these limits produced discernible degradation in the images.

The experimental text-access system

Figure 2 contains a diagram of the text-access system and its connection with the complete INTREX system.[1,9,10] The primary retrieval programs and data storage for the augmented catalog are filed within MIT's general-purpose compatible time-shared computter utility (CTSS). The buffer/controller (B/C) includes processing capability for controlling the operating modes of the catalog consoles, and directing the data flow among consoles, and the text-access system.[3] The processor in the B/C, a Varian Data Machines 620I computer, serves as a systems monitor among the elements of the INTREX system as discussed in References 2 and 3.

The B/C is connected to the text-access-system central station via a 300-bit-per-second half-duplex serial data channel. The terminal equipment is designed to interface with a standard dataphone to permit the use of a common carrier. Because of the rather specialized and limited data requirements of the text-access system, computer words, not characters, are transmitted to the text-access central station.

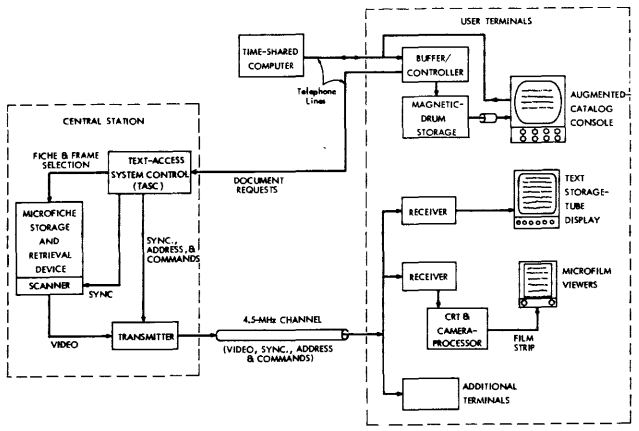


Figure 2—Text-access system

Software

A flexible software package has been developed for the 620I to control data flow efficiently and to provide linkage to subprograms that provide special services for the user consoles. As the 620I dedicated to INTREX has only 4K of core memory, the monitor system will have the capability of storing subprograms on a portion of the 128 track magnetic drum which is used to refresh the console displays.

It is anticipated that requests for text access will be a result of a catalog search. After retrieving a document title the user may wish to check on its relevancy by seeing the first page or to read the entire article. This desire will be indicated by activating a button on the augmented-catalog console which will cause the 620I to fetch the text-access subprogram into core.

This subprogram will permit the 620I to carry on a dialog with the user during which he will identify the document he wishes displayed. Initially the user will have to type the access number which he retrieved on a previous catalog search, when the 620I asks for it. This access number identifies the microfiche and frame numbers that locate the document in the text-access files. This rather laborious procedure will be changed as the software interface between the buffer/ controller and CTSS is more fully developed. In the future, CTSS will tag the document access number for the 620I so that the user need only identify this document either by typing its title or pointing at its title as displayed on the augmented-catalog output with a light pen or cursor. The 6201 would then associate the document with the correct access number.

After identifying the document, the user will be asked whether he desired to see the first page of the article displayed on the direct-view storage-tube display terminal or whether he would prefer a film copy of the entire document. This choice will be indicated by typing C for copy, D for display, or pushing a programmable button.[3]

If the display option is taken, the action will shift to the text-access terminal where the first page of the document will appear in a few seconds. The 620I will remember the access number and page number of the text displayed on the text-access screen. To view succeeding pages or a magnified image of a sector of the current page the user need only push illuminated buttons located adjacent to the text display unit. These buttons, labeled PAGE-FORWARD, PAGE-BACKWARD, REDISPLAY-SAME-PAGE, MAGNIFY, DISPLAY-CHOSEN-SECTOR, and SECTOR-POSITION (a matrix of 9 buttons), are illuminated in a programmed sequence to guide the infrequent user. They are connected through the augmented-catalog console to the buffer/controller.

Currently the 620I maintains a queuing routine for the time-shared central station of the text-access system. Even with only one text-display console a queue is likely to form as the user requests film copies of several documents, while continuing to inspect pages of others on the storage-tube display. With several user terminals operating, a copy queue and a display queue will be formed. At first, priority is allotted to the display queue as it is thought that changing pages on the storage display should be a very rapid operation to facilitate browsing.

As the queues are formed, the 620I checks to see that the access number, whether typed by the user or retrieved from CTSS is valid. This procedure avoids sending erroneous data to the text-access system which could result in delay in the operation of the text displays.

Text-access central station

The central station contains the document store, an automatic retrieval device, a flying-spot scanner, and control logic.

Much attention has been given to the problem of communication between the 620I computer and the Text Access-System Control logic (TASC logic). Flexible operation is desirable with a minimum of information transmission between the devices. The text-access central station is treated as an output device and the TASC logic is a special purpose processor that actually operates the unit. An output request from the 620I consists of one, two, or three 16-bit computer words that contain the fiche number, a frame number and certain procedural data.[2] The format of these computer words was chosen to minimize the number of words required per request and yet not overly complicate the TASC logic. The TASC logic also sends 16-bit status messages back to the 620I after the completion of every text-access request or in the case of a minor malfunction. A more detailed description of the design and construction of the TASC logic is found in Reference 2.

The microfiche-storage-and-retrieval device

The microfiche-storage-and-retrieval device is a Houston/Fearless Compact Automatic Retrieval Device (CARD) modified to be coupled to a flying-spot scanner and to enable it to be controlled by electrical inputs. The basic CARD unit stores up to 750 microfiche with access times of less than five seconds to any microfiche and one second to any frame on the retrieved fiche. Using the COSATI formatted microfiche, approximately 45,000 pages may be stored on the 750 fiche.

Although systems are available that will randomly access any one of 10^5 fiche, no automatic storage and retrieval device is commercially available that will adequately store and access the contents of a complete university library containing more than 10^6 volumes. The Houston/Fearless machine was chosen for the initial INTREX experiments because it economically met the requirements necessary for accessing the documents in the INTREX collection and its response time for a limited number of user terminals is quick enough to test the principle of on-line browsing. It is anticipated that the partial specifications for a device capable of storing a much larger collection will be among the results of the INTREX experiments.

The transmission subsystem

The transmission subsystem links the user terminals to the central station via a unidirectional coaxial line. The coaxial line is time shared among the user terminals; therefore provision is made for uniquely addressing each terminal. Synchronizing pulses for the entire system are generated by an oscillator contained in the transmitter section of the TASC logic.

The output of the oscillator is divided to produce a basic clock frequency of 280 kHz which is further divided by programmable counters to produce the horizontal-synchronizing pulses. The number of scan lines is determined by a programmable line counter that counts these pulses. The programmable counters provide the flexibility for switching automatically between user terminals having differing scanning parameters.

The horizontal-synchronizing pulses are transmitted continuously to the user terminal. The time between pulses may be occupied by no signal, or an analog video signal corresponding to one line of an image, or a sequence of pulses representing a digital word. Each frame of video is preceded by two 16-bit digital words and followed by one or two digital words. The digital messages are used to control the user terminals. A 6-bit address code is assigned to each user terminal; thus, commands may be sent to any one of 64 possible terminals while others remain idle. The standard ASCII seven-bit code has been chosen for the commands so that a full character set is available. Commands that operate the receiver terminals such as ERASE, ADVANCE-FILM, and BEGIN-VERTICAL-SWEEP were chosen from the ASCII control characters.

The synchronizing signals, digital address and digital command signals, and the analog video signals are combined in a line driver for transmission to the user terminals over coaxial cable. Presently base band transmission is used but the signal format need not be changed if it is decided to utilize a modulated-carrier type of transmission.

A matched filter and threshold comparator in the receiver of each user terminal separates the digital codes from the analog video signal. The decoder functions as an address detector and interpreter of the digital commands. A feature of the video circuitry is the automatic gain control which compensates for slow variations in the system gain which might result from temperature variations and/or changes in the separation distances between transmitter and receiver.

Textual-image displays

Library users are accustomed to the traditional forms of textual images, such as books and journals, which provide high-quality images and many other features including portability, browsing capability, gray scale, color, etc. Many of these features are difficult to achieve in a remote display and a completely satisfactory device for displaying textual images is not yet available. Although the text-access display may suffer from comparison with the traditional textual forms, this is offset by providing guaranteed, rapid access to full text at a console conveniently located near the researcher's working area.

The detailed requirements for the Intrex displays have been reported previously and are reviewed in a companion paper at this conference. The single-frame-transmission feature of the text-access system, requiring image storage at the user terminals, has a significant effect on the types of displays that are suitable for this system. The video signal could be stored and used to generate a refreshed CRT display. However, the bandwidth in excess of 60 MHz required for a flicker-free display with adequate resolution is a serious obstacle to this approach. High-order interlaced scanning reduces the bandwidth requirements somewhat, but some brief experiments performed by our group with pseudo-random scanning indicate that sufficient bandwidth reduction cannot be achieved to make the refreshed textual display practical.

Two types of display terminals are included in the initial system. One uses an electronic-storage tube and the other uses 35-mm film for image storage. The cathode-ray storage tube is an erasable, soft-copy display and the film terminal provides a form of hard copy. The soft-copy display permits rapid access to text because the image requires no processing. A browsing capability requires a response time of at most a few seconds between pages. An erasable storage medium is potentially more economical in cost per page because there is no material expenditure for each display request. If the text is always available within seconds from a central store, it is expected that much of the need for hard copy will be eliminated. Unfortunately, there is no existing transient-display device with the resolution and writing speed required for the text access system, but the direct-view electronic-storage tube comes closest among the currently available devices.

Adequate resolution is achieved with the microfilm-facsimile terminal. The output of this terminal is a 35-mm film strip which is automatically processed in approximately one minute and read with the aid of microfilm viewers.

Storage-tube display

A block diagram of the storage-tube display terminal is presented as Figure 3. The terminal consists of two main components, the Tektronix type-611 Storage-Display Unit and the electronics for controlling the display and user inputs. It is designed to be self-sufficient in that it requires no external power supplies.

The storage tube terminal is located adjacent to the augmented-catalog console and is intended to provide a quick-look at full text as a supplement to the catalog searching operations. The limiting resolution of the Tektronix eleven-inch storage-display unit is approximately 400-line pairs in its long dimension which is considerably less than the 1000-cycles/page that the image-transmission experiments showed to be a minimum for high-quality textual images. The lack of resolution and gray-scale capability results in an information loss particularly for small symbols for characters, and in pictorial material. In addition the brightness of the Tektronix 611 is marginal for viewing in a well-lighted room. However, the display is not intended for prolonged reading or for detailed text, but it is appropriate for evaluating the usefulness of a

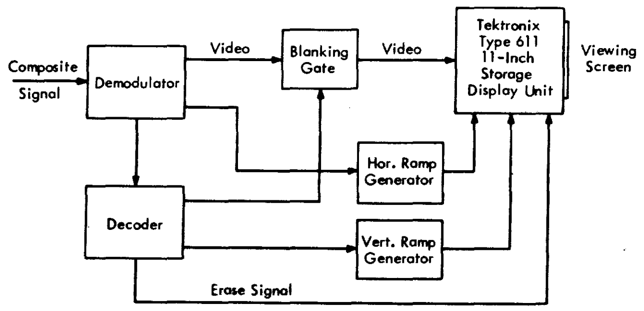


Figure 3—Storage tube display terminal

soft-copy, stored display as part of the text-access experiments.

In addition to the input devices at the catalog console, a number of illuminated switches are located at the storage-tube display and are connected to the console. Two of these, PAGE-FORWARD and PAGE-BACKWARD, enable the user to request the following or preceding page in a document by pushing a single button.

A third function switch is used to initiate the magnify mode which is designed to compensate, in part, for the limited resolution of this display. In this mode, an illuminated rectangle with dimensions approximately one-half the full page size appears as an overview on the display. The rectangle outlines the page sector to be magnified and can be moved to any one of nine positions by means of a pushbutton matrix at the display. When the re-display button is pushed, the quarter-page sector outlined by the rectangle is scanned and transmitted. This gives a factor-of-two magnification of that page sector and improves the legibility of small characters which might not be recognizable on the full-page display.

Several indicator lights are included at the storage-tube terminal to inform the user of the status of his request such as FICHE-NOT-FOUND, LAST-PAGE-OF-DOCUMENT, and REQUEST-IN PROCESS. The pushbuttons and indicators are lighted in a programmed sequence to assist the infrequent user in operating the terminal.

Microfilm-facsimile terminal

The microfilm-facsimile terminal, diagrammed in Figure 4, consists of a high-resolution cathode-ray tube with its associated sweep and focus circuitry, an automatic camera-processor, and control logic required to operate the terminal. On command from the central station, the microfilm-facsimile terminal will reconstitute a page of text on the face of a high-resolution cathode-ray tube from the video signal. The automatic-camera and film-processor unit will record on 35-mm film the image of the displayed text and deliver to the user a fully processed strip of film in a convenient form for viewing in a microfilm reader.

A camera-processor unit that satisfactorily met the INTREX requirements was not found to be commercially available. The camera-processor pictured in Figure 5 was manufactured by attaching a modified Kodak film unit to a GAF automatic film processor.[2] This processor utilizes a horizontal straight-line film transport that is self-threading and accepts short strips of 35-mm film. It should be noted that the film processor was designed for films of 12-inches maximum width, and for a much heavier volume of processing than we anticipate. It appeared after a survey of available film processors that the GAF machine comes closest to meeting our needs, at least on a temporary basis.

Initially the request for a film copy, made by the user at the catalog console will result in a film strip containing the entire text of a journal article if the document contains eight or fewer pages. Longer documents will be filmed in 8- page increments. Provision has been made to allow the user to combine short documents on a single film strip.

Approximately 20 seconds is required to complete the filming of an eight-page document after the receipt of a request by the text-access central station. After the filming of the last page in a sequence, a digital command transmitted from the central station initiates

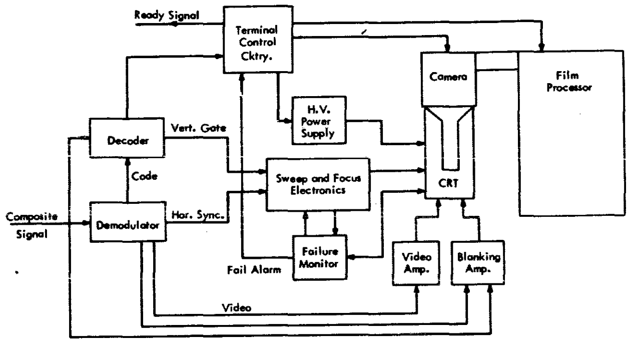


Figure 4—The microfilm-facsimile terminal

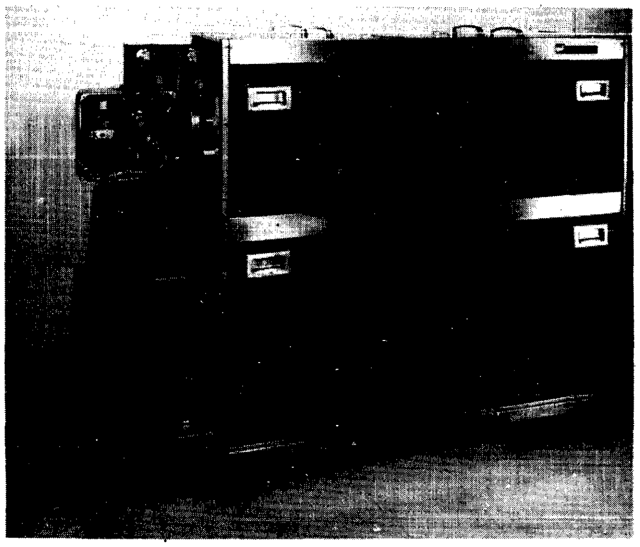


Figure 5—The camera processor

the processing of the film strip which requires approximately 70 seconds.

User acceptance of the microfilm output is largely dependent upon the convenience of handling and viewing the 35-mm film strips. After emerging from the processor, the film strip is inserted into a transparent jacket which facilitates handling and protects the film. A strip along the jacket edge can be written on with pen or pencil for identification purposes. The film in the jacket is inserted into a microfilm viewer for reading.

An Addressograph Multigraph Model 3000 electrostatic copier is being modified such that 8 1/2-by-11-inch paper copies can be made from the 35-mm film images. There is some degradation in the copying process and these copies lose some resolution compared to the 35-mm film image. However, the modified machine will supply, at locations remote from the central store, a traditional form of hard copy that can be read without the need for a viewer.

SUMMARY

A system for providing guaranteed, rapid, and remote access to the full text of the 10,000 documents in the INTREX collection has been described. Consideration of the more general problem of storing and displaying the textual image has indicated the practicality of facsimile image storage on microfilm and of single-frame transmission from a time-shared central station to the user terminals. In the initial INTREX text-access system, photographic images of text are stored on microfiche which are accessed by a computer-controlled storage and retrieval device. Retrieved fiche are automatically positioned in order that the proper frame may be scanned and transmitted as a single frame of video to either of two user terminals. A direct-view storage-tube display unit, placed adjacent to the catalog console, provides rapid access to text although with marginal resolution and brightness. A microfilm-facsimile terminal provides adequate resolution, but the film-processing time and mechanical complexity of the terminal are significant disadvantages. Document requests are entered through the augmented-catalog console. The processor associated with the catalog buffer/controller maintains the queuing and message-formatting algorithms for the text-access system.

The text-access system is currently operating in the laboratory and is part of the INTREX facilities intended for user experiments. Evaluations of these experiments should provide new insights into the library user's requirements for text-access which will lead into the incorporation of new techniques and equipment.

ACKNOWLEDGMENT

The research reported in this paper was conducted at the Electronic Systems Laboratory, Massachusetts Institute of Technology, as part of Project Intrex and was supported through grants to Project Intrex from the Council on Library Resources, Inc., and the Carnegie Corporation.

REFERENCES

1 J F REINTJES
System characteristics of intrex
Proc S J C C 1969

2 Project Intrex Staff
Project intrex semiannual activity report
September 15 1968

3 D R HARING
Computer-driven display facilities for an experimental computer-based library
Proc F J C C 1968

4 C F J OVERHAGE R J HARMAN (editors)
Intrex report of a planning conference on information transfer experiments
M I T Press Cambridge Massachusetts 1965

5 U F GRONEMANN D R KNUDSON S N TEICHER
Remote text access for project intrex
ESL TM-312 July 1967
This report based on a Conference paper presented at the National Microfilm Association Convention Miami Beach Florida April 26–28 1967

6 D R KNUDSON S N TEICHER J F REINTJES U F GRONEMANN
Experimental evaluation of the resolution capabilities of image-transmission systems
Information Display September/October 1968

7 A VAN DAM J C MICHENER
Hardware developments and product announcements
Annual Review of Information Science and Technology Vol 2 1967

8 P BROWN S JONES
Document retrieval and dissemination in libraries and information centers
Annual Review of Information Science and Technology Vol 3 1968

9 D R HARING J K ROBERGE
A combined display for computer-generated data and scanned photographic images
Proc S J C C 1969

10 R S MARCUS P KUGEL R L KUSIK
An experimental computer-stored augmented catalog of professional literature

11 P A CRISMA (editor)
The compatible time-sharing system
M I T Press 1968

A combined display for computer-generated data and scanned photographic images *

by DONALD R. HARING and JAMES K. ROBERGE

Massachusetts Institute of Technology
Cambridge, Massachusetts

BACKGROUND

Project Intrex (**IN**formation **TR**ansfer **EX**periments) is a program of research and experiments intended to provide a foundation for the design of future information-transfer systems. The library of the future is conceived as a computer-based communications network, but at this time we do not know enough details about such a network to design it. Lacking are the necessary experimental facts, especially in the area of user's interaction with such a system. To discover these facts, we want to conduct experiments not only in the laboratory, but above all, in the real-life environment of a useful operating library.[1,2]

The initial efforts of Project Intrex have been concerned with the problems of access—bibliographic access through an augmented library catalog, and access to full text. This paper describes the design of the initial computer-driven display facilities being developed for the Project Intrex experimental computer-based library. To provide further background information, we will first give some details of the augmented library catalog and the text-access system that are being developed.

For a number of reasons, computer-based libraries that service a wide spectrum of users, such as are found in a university, will be faced with operating on two basically different types of data—that which is digitally stored and that which is photographically stored in some microfilm form. The latter will be images of the original full text of the documents contained in the library, whereas the digital data will constitute the augmented catalog of the library from which the library user gleans information about the library data and documents by conducting interactive computer searches.

One of the concerns of Project Intrex is to conduct a series of experiments to determine how the traditional library catalog can be effectively augmented and combined with on-line computer operation to provide users with a more powerful, comprehensive, and useful guide to library resources. Present plans call for augmenting the traditional library catalog in scope, depth, and search means. For example, the augmented catalog will contain entries for reports and individual journal articles as well as the traditional entries for books. Furthermore, in addition to the title and author of each item, such things as the bibliography, an abstract, key words, and key phrases of each item will be included as part of its catalog entry.[2,3] A user will be able to conduct searches on nearly any combination of the data contained in a catalog entry. Present plans also call for providing alphanumeric communications between user and computer by means of a high-speed, flexible display console.[2,4,5]

Another concern of Project Intrex is to conduct a series of experiments in an effort to devise a workable system that will ultimately provide guaranteed, rapid access to the full text of journal articles, books, reports, theses, and other library materials at stations that are remote from the central store. This goal has several implications. Guaranteed accessibility implies that text never leaves its store and is therefore available to users at all times. Availability with minimum delay at remote locations implies transmission of text in

* The research reported here was made possible through the support extended the Massachusetts Institute of Technology, Project Intrex, the Electronic Systems Laboratory, under Contract NSF–C472 from the National Science Foundation and the Advanced Research Projects Agency of the Department of Defense, and under Grant CLR–373 from the Council on Library Resources, Inc.

electrical signal form, except in special, limited situations where physical transmission (perhaps by pneumatic tubes) might be appropriate. Remote accessibility implies more convenient library use, and in addition is a preliminary step toward realization of a network of computer-based libraries coupled together by means of data communication links.[2,6]

In order to conduct meaningful experiments, a small-scale experimental total computer-based library system containing some 10,000 systematically-chosen documents in materials science and engineering is being designed and constructed by Project Intrex.[2] The computer-driven display console discussed here is a part of this system.

Figure 1 illustrates the general organization of the experimental library. The library operates roughly as follows: The digital data and the photographic images are created from the original documents. The former are placed into the storage of the time-shared computer (an *IBM* 7094 system) and the latter are placed into a microfiche storage and retrieval unit (a modified *Houston Fearless* CARD machine). In the original system the stored data are then accessed through augmented-catalog user consoles and text-access user terminals, respectively. This paper is concerned with the description of an experimental display that can accommodate both the augmented-catalog digitally-coded data and the video information generated by scanning (in facsimile fashion) the full-text photographic-image data.

The plan of this paper is to first briefly discuss the display requirements of the Intrex experimental library and then to describe the display techniques that are being developed. Because of the differences between the storage form and content of the catalog (digital, alphanumeric) and the full text (photographic images, graphic), and because of the differences between user interaction with the catalog and full text, we find it convenient first to describe the two displays separately, and then to describe the combined display.

Previous papers and reports have described the Intrex experimental computer-based library, the display requirements, and the separate displays for the augmented-catalog and full-text data.[2–7] These papers should be consulted for further details of these subjects. This paper will concentrate on the features unique to the present display and the details of certain system parameters not discussed previously.

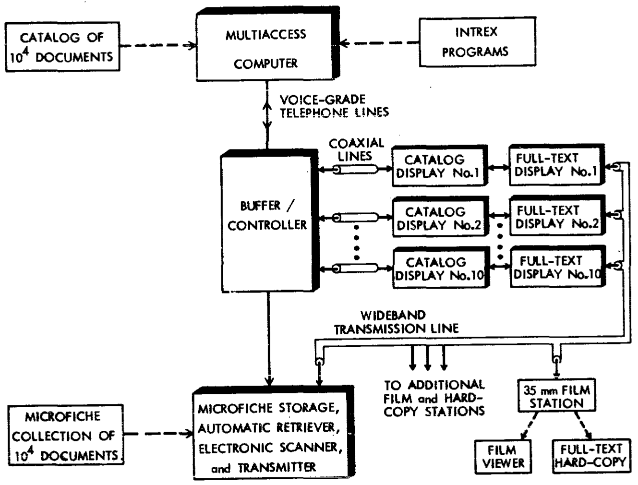


Figure 1—Project Intrex experimental library

Augmented-catalog display requirements

Detailed examination of system requirements, coupled with the general intent of Project Intrex to provide sufficient flexibility for meaningful experimentation with information transfer techniques led to the requirements outlined in this section. A more complete explanation of underlying system requirements is presented in References 2 and 5.

A refreshed display seems preferable to a stored display which can be changed only with a complete new transmission from a remote computer. The use of a refreshed display complicates local memory requirements, but facilitates local editing and permits the use of a light pen for message identification.

A large capacity display which can present all the available data about any given entry is more pleasing than a smaller display which may require viewing several different messages concerning a single document. Similarly, comparison of key words or bibliographies of several documents is simplified by a large capacity display. These considerations, coupled with the resolution capabilities of inexpensive cathode-ray tubes (CRTs) led to the choice of an 1800-character display.

It seems desirable to expand the character set beyond the 96 visible characters of the USASCII set. In particular, the scientific data base anticipated for Intrex experiments makes the addition of Greek letters and more mathematical symbols a necessity. Furthermore, it should be possible to reproduce subscripts and superscripts in a natural and pleasing manner. It also seems advantageous to have an easily alterable character set so that foreign documents can be cataloged in their own alphabet if desired.

There is also the underlying objective of developing a display which is economical to reproduce, since economy is one important factor in the large-scale acceptance of the Intrex concept.

The display tube and deflection technique

Much of the expense of large capacity refreshed alphanumeric displays reflects the cost of the CRT and its deflection electronics, and any meaningful attempt at cost reduction must consider this equipment.

The resolution necessary to display 1800 characters is not demanding. The 500 × 700 line resolution of monitor quality entertainment CRT's allows upwards of 12 resolvable elements in both dimensions of a character and this resolution is sufficient for high-quality characters. A more fundamental difficulty is encountered when the deflection bandwidth normally required for rapid character generation is considered. In order to refresh an 1800 character display at a nominal 60 Hz rate (sufficient to prevent flicker) it is necessary to generate characters in approximately 8.5μs. Two popular generation methods are the selective intensification of a dot matrix or a pattern from a stroke generator. The mechanization of either a 5 × 7 dot matrix generator (which does not produce particularly attractive characters) or a 20-segment stroke generator requires deflection-system settling time on the order of 0.1μsec. The cost of this type of deflection system for a large screen CRT is high at the present time.

An alternative is the use of a TV type scan which is relatively narrow band. However, a conflicting constraint is introduced by the memory. Economics dictate the use of a serial memory such as a magnetic drum or a delay line for display maintenance, and the cost of scan conversion equipment for such a memory appears to outweigh the advantages of the TV scan.

The scan selected for the alphanumeric display consists of two parts. The basic raster illustrated in Figure 2 is generated and applied to deflection amplifiers which drive a conventional deflection yolk. The raster pattern provides for 31 lines of 56 characters each, refreshed at a 57.5 Hz rate. (Actual timing allows for 32 lines of 64 characters each, but one line per raster and 8 character spaces per line are reserved for retrace.) It is important to note that the deflection bandwidth requirements for this pattern are significantly less demanding than those of a conventional TV raster.

A 600-kHz sinusoidal signal which provides a peak-to-peak deflection amplitude slightly in excess of the maximum anticipated character height is added to the vertical deflection signal. This sinusoidal signal

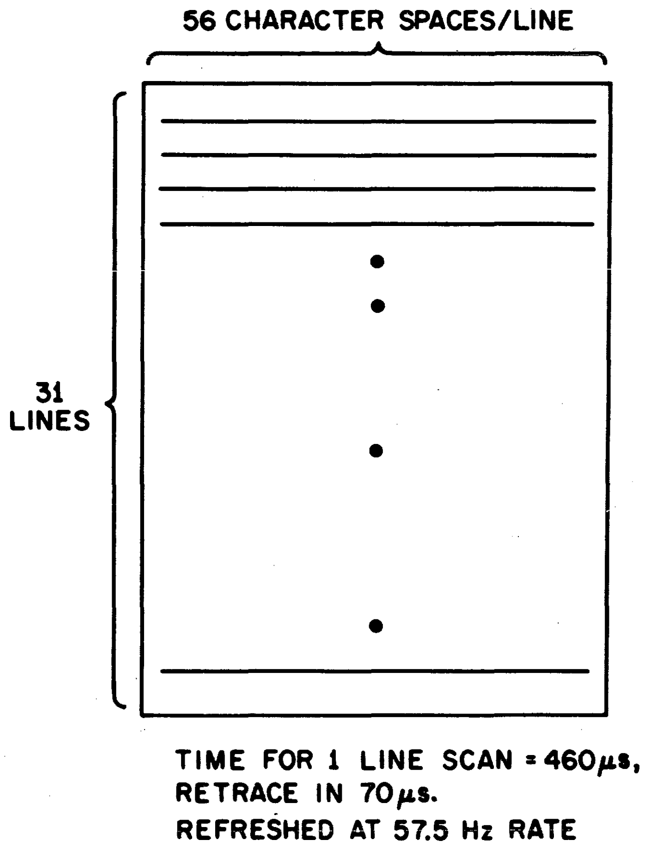


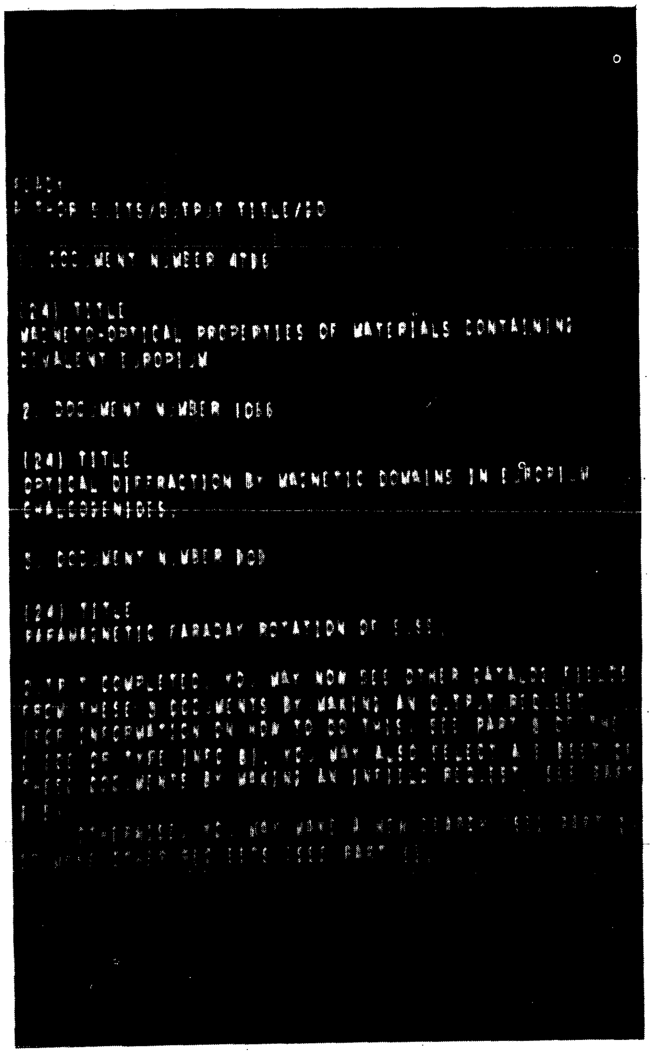
Figure 2—Basic raster pattern

forms the scan pattern on which characters are generated by appropriate intensity modulation. (The method used to generate video information is described in the following section.) The advantage of the sinusoidal scan is the limited bandwidth necessary to reproduce a single-frequency sinusoid. In practice, the 600 kHz signal is amplified by a tuned narrow-band amplifier with a transformer-coupled output. The secondary of this transformer is placed in series with the vertical deflection coil and the output of its deflection amplifier to provide the scan signal.

The linear segments of the 600-kHz signal provide 10 scan lines per character space. In order to further improve the quality of the characters, the phase of the scanning sinusoid is changed by 180° on alternate frames. This process is analogous to the interlacing used in TV transmission and results in improved resolution without increasing required video bandwidth.

The greatest demand on deflection bandwidth arises when superscripts or subscripts are generated. In order to display such a character, a code indicating the desired operation is placed into the memory which contains the display data. The code adds a signal corresponding to ± ¼ or ± ½ of the nominal vertical spacing to the vertical deflection signal. Simultaneously, a signal corresponding to the spacing between characters is subtracted from the horizontal sweep. This backspace technique allows inclusion of the control code without creating blanks in the display. However the settling time of both deflection amplifiers must be less than the time normally used for a single character (8.5μs) for the method to be effective. Following a time delay corresponding to one character, the elevated or depressed character is generated normally and displayed in its correct location.

The first augmented catalog display system used a $15 CRT from a TV set as the display element. The deflection yolk was the one designed for use with the tube, modified by reducing the number of windings on the vertical axis to approximately 20 percent of its initial value. Figure 3 illustrates the quality of the characters produced by this inexpensive display system. A moderate improvement in character fidelity, particularly at the edges of the display, has been obtained using a monitor quality tube and an improved deflection yolk. These improved components add approximately $75 to the cost of the equipment.



```
[illegible]
[illegible] ITS/OUTPUT TITLE/[illegible]

1. DOCUMENT NUMBER 4786

(24) TITLE
MAGNETO-OPTICAL PROPERTIES OF MATERIALS CONTAINING
DIVALENT EUROPIUM

2. DOCUMENT NUMBER 1066

(24) TITLE
OPTICAL DIFFRACTION BY MAGNETIC DOMAINS IN EUROPIUM
CHALCOGENIDES.

3. DOCUMENT NUMBER 909

(24) TITLE
PARAMAGNETIC FARADAY ROTATION OF EUSE.

[illegible] COMPLETED. YOU MAY NOW SEE OTHER CATALOG FIELDS
FROM THESE 3 DOCUMENTS BY MAKING AN OUTPUT REQUEST.
(FOR INFORMATION ON HOW TO DO THIS, SEE PART 8 OF THE
[illegible] OR TYPE INFO 8). YOU MAY ALSO SELECT A [illegible]
THESE DOCUMENTS BY MAKING AN INFILE REQUEST. (SEE PART
[illegible]
[illegible] YOU MAY MAKE A NEW SEARCH [illegible]
[illegible] PART [illegible]
```

Figure 3—A display of the output of the character generator

The character generator

The choice of scan described in the preceding section imposes constraints on the type of character generator used in conjunction with the display tube. While use of a digital read-only memory for character generation is possible, this type of memory does not readily produce video information for a sinusoidal scan pattern. An extremely straightforward system is obtained if a flying-spot scanner is used for character generation. Figure 4 illustrates the major components of a character generator using this principle.

The character set is stored on a photographic negative. (A picture of the character mask currently used is shown in Figure 5.) This mask is located in front of a small electrostatically-deflected CRT and can be replaced easily to alter the character set. In order to generate a character, the beam of the scanner CRT is positioned to the left of the desired character. Beam positioning is controlled by digital-to-analog converters in response to the code for the selected character. The character is then scanned by applying a ramp as the horizontal deflection signal and a sinusoid derived from the display-tube scan as the vertical deflection signal. Light which shines through the character mask is detected by a photomultiplier tube to produce the video signal.

In order to achieve resolution in the vertical direction

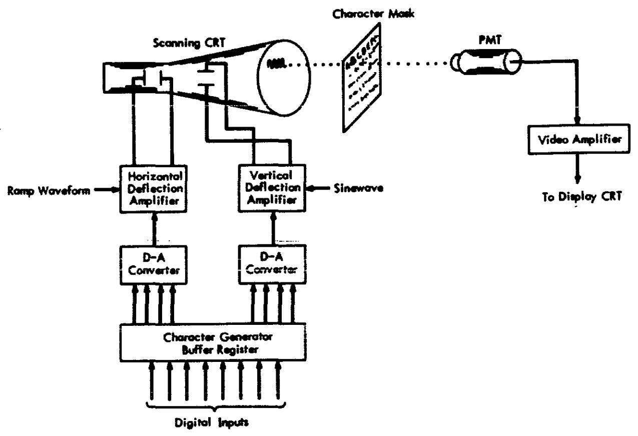


Figure 4—Flying-spot-scanner character generator

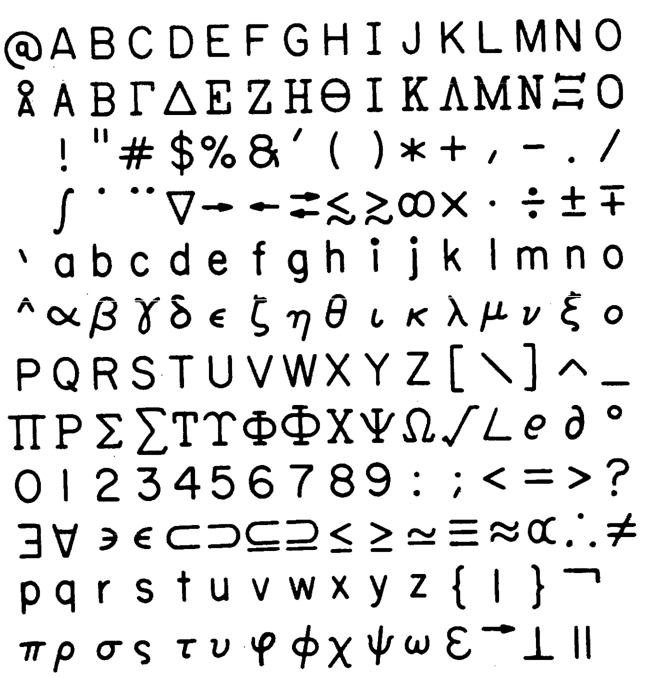


Figure 5—Character mask

comparable to that produced horizontally, it is necessary to generate blanking pulses which are on the order of 30 nanoseconds (ns) wide. The rise time of the photomultiplier tube is less than 5 ns and a blanking amplifier which switches the grid of the display CRT 40 volts in approximately 5 ns was designed for this application. The limiting factor is the decay time of the P 16 phosphor used in the scanner CRT, and this limitation dictated the use of a 600-kHz scan rather than a higher value.

The monoscope tube[8] offers an alternative to the flying-spot scanner character generator described in this section. Monoscopes are available with 96 character targets, and two such units can share common electronics to provide the desired 192 character set. A disadvantage is the need to change monoscopes to alter the character set.

A more detailed description of the Intrex character generator is presented in Reference 9.

The text-access display requirements

The differences between the augmented-catalog display requirements discussed in the previous section and the text-access display requirements are due to the differences of the storage medium and types of user interaction with the stored data. The catalog data are digitally stored in a file that is attached to the time-shared computer and is accessed through and processed by that computer. The library user can conduct computer searches on this data and modify his copy of the data. Thus, if he wants hard-copy of this data he can have the data organized in any desired form. On the other hand, the full-text data are photographic images that are accessed through a mechanical storage and retrieval unit that is attached to the display facility. This data does not pass through a computer. Its form cannot be modified (except for magnification) and the user cannot conduct computer searches on the data. The only access to full-text data is by call number. Once the data is obtained, the user can read it, can ask for other pages of the same document or ask for a magnification of portions of the page of the document. In short, the dynamics and the dialog between man and computer are considerably different between the two display facilities.

With the original text stored in image form, the problem of remote access to text becomes that of reproducing a high-quality image at a remote point. This usually implies scanning the stored images. Our experiments have shown that at least 2000 scan lines, with a total system on-axis limiting resolution of 1000 cycles per frame height from scanner to viewer, appear to be needed for high-quality remote reproduction of microfilmed technical documents (18-to-1 reduction ratio) having average quality and containing the subscripts, superscripts, and mathematical symbols that frequently appear in these texts.[2] A two-column page with approximately 70 lines per column and an average of eight to ten words per line is considered to be a "typical" page.

Closed-circuit TV requires high bandwidth (80 MHz) to achieve the desired resolution of 2000 lines per page. Also, because the image is continually refreshed, a separate scanner and transmission line would be required for each simultaneous user. Hence, a facsimile-like system is used in which each text page is scanned and transmitted only once, and the information is captured and stored at the receiver for transient viewing (soft copy) or printing (hard copy). This organization permits a tradeoff between signal bandwidth and transmission time, and also permits time multiplexing of the microfiche on which the image is stored, the microfiche storage and retrieval device, the scanner and the transmission line to serve a number of users. For example, with a transmission time of ½ a second per page, signal bandwidth for a 2000-line scan is reduced to about 4.5 MHz (standard TV channel bandwidth) and a single text-access system can perhaps service 20 to 30 receivers, assuming that users would request new pages at 10 to 15 second intervals.

The soft-copy display appears to be the more attractive because it more closely approaches the capability of providing immediate remote text access and is potentially the least expensive to operate. Assuming a facsimile-type scan, the following list summarizes the specifications of an ideal soft-copy medium for single-page, full-text display.[2,10,11]

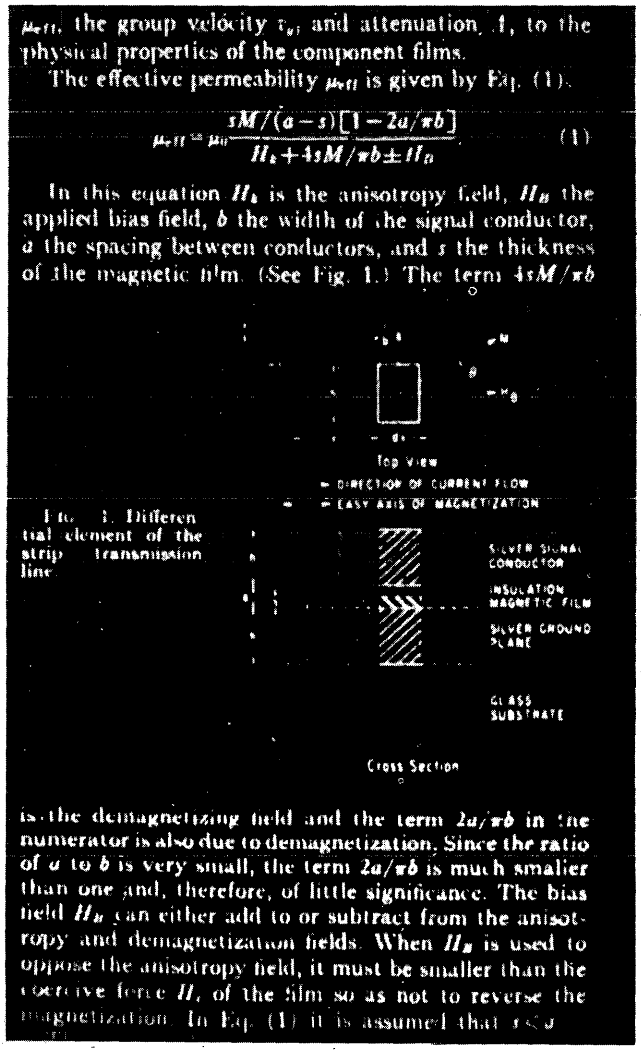
1. *Size*: Display area should be approximately 8½ by 11 inches.
2. *Resolution*: Minimum limiting resolution for any portion of the display area should be equivalent to more than 1000 cycles per frame height.
3. *Brightness and Contrast Ratio*: The viewable image should be sufficiently bright to be comfortably seen in a room lighted for reading (60 to 160 foot-lamberts). The image should have a contrast ratio of at least 15 to 1, a sharpness (or acceptance) comparable to that of a printed page.
4. *Storage Time*: At least five minutes
5. *Viewing Position*: Freedom from constraints on the user viewing position and freedom to position and orient the display itself.
6. *Gray Scale*: Sufficient to faithfully reproduce high-quality black-and-white photographs. (eight gray levels)
7. *Speed*: Sufficient speed so that a complete medium erase (or replace) plus write requires approximately one second, with the time nearly equally divided between the two operations.
8. *Color*: Sufficient color range to faithfully reproduce color photographs.
9. *Cost*: Cost of ownership (including cost of storage medium) such that the display of a page of text be significantly less than a hard copy of that page. (In the order of 0.01 cent per page.)

Unfortunately, no presently available medium fully satisfies these requirements. However, the Tektronix Type 611 eleven-inch storage-tube display unit does afford limited capabilities.

The storage-tube display

An evaluation of the Tektronix display unit indicates that it can serve as a satisfactory *experimental* soft-copy display. As regards our list of specifications for a soft-copy medium, we make the following list of observations.

1. *Size*: The display area is 6⅜ inches by 8¼ inches. This is adequate but preferably should be somewhat larger.



μ_{eff}, the group velocity v_g, and attenuation, A, to the physical properties of the component films.

The effective permeability μ_{eff} is given by Eq. (1).

$$\mu_{eff} = \mu_0 \frac{sM/(a-s)[1-2a/\pi b]}{H_k + 4sM/\pi b \pm H_B} \qquad (1)$$

In this equation H_k is the anisotropy field, H_B the applied bias field, b the width of the signal conductor, a the spacing between conductors, and s the thickness of the magnetic film. (See Fig. 1.) The term $4sM/\pi b$

FIG. 1. Differential element of the strip transmission line.

is the demagnetizing field and the term $2a/\pi b$ in the numerator is also due to demagnetization. Since the ratio of a to b is very small, the term $2a/\pi b$ is much smaller than one and, therefore, of little significance. The bias field H_B can either add to or subtract from the anisotropy and demagnetization fields. When H_B is used to oppose the anisotropy field, it must be smaller than the coercive force H_c of the film so as not to reverse the magnetization. In Eq. (1) it is assumed that $s \ll a$

Figure 6—Photograph of transmitted image as displayed on a Tektronix storage tube

2. *Resolution*: Specified to be 800 cycles in the large dimension. This is marginal, especially for perception of poor-quality print or small symbols and characters. Figure 6 shows typical results obtained. (Note: There is some degradation due to the photographic processes.) It is estimated that an improvement of approximately 25 percent in resolution is required.
3. *Brightness and Contrast Ratio*: The stored luminescence of the storage tube is 3-foot lamberts and operating contrast ratio of 2:1. Hence, the general room lighting must be at a lower level than normally used for reading.
4. *Storage Time*: Once an image is stored on the tube, it remains until erased.
5. *Viewing Position*: The user must be relatively close to the unit. He has a viewing angle typical

of most CRT's. Placed in the proper mounting, the position of the unit can be adjusted to a limited extent. The unit is not readily portable.

6. *Gray Scale*: The Tektronix storage tube has only two light levels. Thus, the usual television reproduction of black-and-white photographs is not possible.
7. *Speed*: Erase time is less than 0.5 seconds and full-screen write time is approximately 2.5 seconds. These times are acceptable in the experimental system.
8. *Color*: The Tektronix storage tube is monochromatic.
9. *Cost*: Because of the high cost of the storage tube and its relatively short life (approximately 1000 hours) in the storage mode, the cost per displayed page is in the order of pennies per page. Although too costly in an operating system, this is acceptable in the experimental system.

Operation of the Type 611 as a facsimile display terminal is straightforward. The self-contained amplifiers for the three axes have sufficient bandwidth for the vertical and horizontal scanning signals and the video signal. To match the erase and writing speed of the storage tube, the microfiche scanner completes a 2000-line scan in 4 seconds. These signals are transmitted to the Type 611 unit from the microfiche scanner via a coaxial cable. The signal format on the coaxial cable resembles a standard television signal with vertical and horizontal synchronizing pulses added to the video signals. Since a bus system is used on the coaxial cable such that many display terminals can be serviced by the same cable, digital control signals are added to the combined video and synchronizing signal in the time slot immediately following the horizontal synchronizing pulse (during storage tube retrace time) to individually command the various terminals. The digital control signal is in the form of a 16-bit binary number, with the first seven bits being the address of the terminal and the last nine bits being the command. By this mechanism, only the display terminal that is addressed responds to the video signals and other control signals.

To accommodate the control functions sent on the coaxial cable, each Type 611 display unit has a black box added to complete a text-access soft-copy display terminal. Specific details of the transmitted signals and the black box design can be found in Project Intrex Semiannual Activity Report dated 15 March 1968.[2]

To overcome the resolution limitation pointed to above, an enlarged version of any one of nine overlapping portions of a page of text can be requested. The portion of the page to be enlarged is identified by a rectangle that is displayed on the storage-tube screen in the non-store mode and whose position is under control of the user.

A combined display unit

Previous sections of this paper have indicated how a CRT with limited deflection bandwidth is used for a large-capacity display of digitally coded alphanumeric characters and how a Tektronix Type 611 Storage Display Unit is used for soft-copy display of photographically-stored material. The soft-copy display unit is intended for operation in conjunction with the augmented-catalog console, since it provides rapid and guaranteed access to items which have been discovered by means of a catalog search. Obvious economics result if the two functions can share a common display element.

The Type 611 can serve as the soft-copy text-access display without modification. It would also be possible to use this unit for the display of digitally coded data by combining stored-mode operation with an appropriate character generator. There are at least two disadvantages to this approach. First, certain functions such as selective erasure and light-pen identification which are important to the operation of the augmented-catalog console, are not possible with a stored display. Second, the storage tube is expensive, and its life when operated in the storage mode is limited to approximately 1000 hours. However, the tube can provide a normal (non-stored) display, and its lifetime is comparable to that of conventional tubes when used in this mode.

The Type 611 can provide horizontal full-screen deflection in less than 60μs, and a small-signal settling time under 5μs. Furthermore, the resolution of the tube exceeds that required to display 2000 high-quality characters. Note that the bandwidth and resolution capability of the Storage Display Unit are comparable to those required for the refreshed alphanumeric display described in an earlier section. This consideration led to the decision to use the Type 611 in a storage mode for soft-copy text display and, by slightly modifying the unit, to operate it as a refreshed display system for catalog information. The only modifications which must be made to accommodate the refreshed display are to include provision for the sinusoidal scan signal and to improve the response time of the blanking amplifier.

A transformer was added in series with the vertical deflection amplifier output to provide the required scan signal. Since the self-inductance of the secondary of this transformer is small compared to the inductance of the vertical deflection coil (1 mH), this additional element does not alter the performance of the vertical deflection system. A new blanking amplifier was also designed for the Type 611. This amplifier switches between ground and a level determined by the setting of intensity controls in approximately 5 ns.

SUMMARY

Two basically different display functions required by a computer-based library have been discussed and a single combined CRT display that provides these functions has been described. The display is based on a Tektronix Type 611 Storage Display Unit. The first function it must accommodate is the output of a character generator driven by the digitally-coded library catalog data. By employing a character generator that requires a relatively narrow deflection bandwidth, only minor modification to the Type 611 is required. When operating with the catalog data the storage tube is used in non-store, refresh mode to provide quick-response man-machine interaction and to increase the lifetime of the tube.

The second function the display must accommodate is the video information generated by scanning the full-text photographic-image data. When operating with the full-text data no modification to the Type 611 is required, and the storage tube is used in store mode to provide a large amount of visual information without external storage.

In addition to the display, a new character generator was described. It is basically a flying spot scanner without lenses. A set of nearly 200 characters is defined by a changeable photographic mask. High quality characters are produced with relatively narrow deflection bandwidths.

REFERENCES

1 C F J OVERHAGE R J HARMAN (editors)
Intrex report of a planning conference on information transfer experiments
The M I T Press Cambridge Massachusetts 1965

2 *Project intrex semiannual activity reports*
M I T Project Intrex Cambridge Massachusetts
March 15 1967 September 15 1967 March 15 1968 and September 15 1968

3 A R BENENFELD
Generation and encoding of the project intrex augmented-catalog data base
Proc 6th Annual Clinic on Library Applications of Data Processing University of Illinois Urbana Illinois May 7 1968

4 D R HARING J K ROBERGE
The augmented-catalog console for project intrex part I
M I T Electronic Systems Laboratory Report ESL-TM-323 October 1967
Presented at the IEEE 1967 Lake Arrowhead Workshop on Advanced Computer Peripherals at Lake Arrowhead California August 25-27 1967

5 D R HARING
A display console for an experimental computer-based augmented library catalog
Proc A C M National Conference and Exposition Las Vegas Nevada August 27-29 1968 35-43

6 U GRONEMANN S TEICHER D KNUDSON
Remote text access for project intrex
M I T Electronic Systems Laboratory Report ESL-TM-312 July 1967
Presented at the National Microfilm Association Conference at Miami Beach Florida April 26-28 1967

7 D R HARING
Computer-driven display facilities for an experimental computer-based library
Proc F J C C 1968 December 9-11 1968 255-265

8 *Technical information bulletin*
Type CK1414 SYMBOLRAY Character Generating Cathode Ray Tube by Raytheon Components Division Industrial Components Operation 465 Centre Street Quincy Massachusetts April 15 1966

9 P F Mc KENZIE
A flying spot scanner character generator
SM Thesis M I T February 1969

10 C T MORGAN J S COOK III A CHAPANIS M W LUND
Human engineering guide to equipment design
McGraw-Hill New York 1963

11 H H POOLE
Fundamentals of display systems
Spartan Books Washington D C 1966

A study of multiaccess computer communications

by P. E. JACKSON and CHARLES D. STUBBS

Bell Telephone Laboratories, Incorporated
Holmdel, New Jersey

INTRODUCTION

The communications characteristics of multiaccess* computing are generating new needs for communications. The results of a study of multiaccess computer communications are the topic of this paper. The analyses made are based on a model of the user-computer interactive process that is described and on data that were collected from operating computer systems. Insight into the performance of multiaccess computer systems can be gleaned from these analyses. In this paper emphasis is placed on *communications considerations*. For this reason, the conclusions presented deal with the characteristics of communications systems and services appropriate for multiaccess computer systems.

The problem

Digital computers requiring communications with remote terminals exhibit a set of communications needs which, in some respects, are different from those of both voice traffic and other record communications. It is important for the providers of data communications to have an understanding of the broad characteristics of this communication process so that new, more appropriate offerings can be designed to satisfy these needs.

Previous studies** by the manufacturers and providers of multiaccess computer systems have begun to characterize both the computer systems and their users. The principal interest of these studies, however, has been computer and/or user performance rather than data communciations.

There are several reasons why the quantitative characterization of the communications process is timely but intricate. First, multiaccess computing is still in its infancy. Therefore, computer system design is going through a trial and error process with a high rate of change of system characteristics. Lacking a unified, well-tested body of technical knowledge applicable to the problems of multiaccess computing, systems designers have been led to heuristic solutions to system organization. Certain specific problems such as scheduling algorithms for single and multiple central processors have been studied in detail. No intensive, overall, general system studies, however, have been reported with the constraints of total cost minimization including the effects of system characteristics on communications costs and human factors such as reduction in efficiency due to long turn-around times.

Second, the rate of change of the size of the user community, the number of systems in operation, and the introduction of new equipment and operating systems is high. In fact, most systems are changing so rapidly that a detailed characterization of any one will probably be outdated before it is completed. The insight to be gained from such studies, however, far outweighs the drawback of obsolescence. Indeed, this situation calls for continued study and review.

Third, the applications of time-sharing are diverse. Where one of the parties in the transaction is a person, uses range from inquiry-response systems with short call durations of a minute or less, to scientific problem-solving and certain types of business information systems with call durations of 10 to 30 minutes, to computer aided learning with long call durations of one to two hours or more. Where the transaction involves an automatic terminal such as a telemetry device, call durations may be measured in milliseconds. Also, the volume of information exchanged in a computer-to-computer or computer-to-data-logger interaction varies widely from a small number of bits in polling, meter

* The word "multiaccess" is chosen to avoid confusion over the use of the word "time-shared" which is often used synonymously but which has a specialized meaning in some contexts.

** For example, see References 1, 2, 3 and 4.

reading and some banking and credit services, to a large number of bits in CRT displays, information retrieval and file manipulation. The speed of transmission is wide-ranging from the low bit rates of supervisory and control terminals to megabits per second for CRT displays.

Fourth, the data required for such studies are microscopic in nature. Unlike voice traffic, which can be characterized by measures of holding times, arrival rates and other parameters independent of a call's content, the characterization of calls to a computer requires some information about a call's content, e.g., timing information interrelating the transmission times of data characters is essential for the design of an efficient time division data multiplexer. An additional factor is that some of the desired statistics on these data have very skewed distributions. Thus, large data samples are required. The implications of these considerations upon our study are that:

a. new data gathering procedures and equipment are needed,
b. data analysis procedures must be capable of handling very large quantities of data,[5]
c. legal, ethical, and business requirements related to communications and computing privacy must be satisfied.

The problem, then, is to provide communications services to a rapidly growing market of multiaccess computer systems and their terminals. These exhibit diverse and changing communications requirements. The study described below is directed at this problem.

The modus operandi for this study is an in-depth analysis of selected multiaccess computer communications systems. The subset of system types chosen for detailed study is composed of computer service providers whose systems are representative of multiaccess computer installations. Besides representativeness, additional prerequisites for the choice of a system to study were that the use of multiaccess computing be advanced, and that the provider of the particular system be knowledgeable in the communications area. By "advanced in multiaccess usage," we mean that the system be fully operational on a daily basis with the initial break-in period accomplished. A final prerequisite for inclusion in the study is the willingness of the computer service provider to participate in the study.*

* Part of the study reported herein involved the collection of data from three operating multiaccess computer systems. In every case these data were obtained on the premises of the computer service provider and with this full permission and cooperation. To ensure the privacy of the three systems under discussion, however, they are not identified by name.

To ensure that a cross section of on-line systems was included in the study, the characteristics of such systems were classified as shown in Table I.

Table I—*Classification characteristics for multiaccess computer systems for communications study*

1. Computer Type
2. I/O Device Type
3. Loading (Number of Simultaneous Users)
4. User's Applications
5. User Community (In-House or Utility)
6. Error Control (e.g., Echoplex)
7. Holding Time

In the table, by "computer type" we mean the manufacturer and model number of the central processor and the system configuration, i.e., whether or not a separate communications computer is used. Not all models of all manufacturers can be covered, but at least two large manufacturers were included for each application. I/O device types include teletypewriter-like terminals and TOUCH-TONE® telephones. Loading is the average percentage of ports that are active. User's applications include scientific and business programming, inquiry-response systems, extended file retrieval and maintenance, message switching and mixtures of these. Both in-house and utility systems were included. Error control includes systems which retransmit each character back to the terminal (Echoplex) and those which do not. The systems selected for examination include short holding time systems with average call durations on the order of one or two minutes or less and long holding time systems with average holding times of 20 to 30 minutes.

From the systems selected for detailed analyses, measurements of three different categories were obtained. The first category included telephone facilities measurements such as occupancies and overflow counts on computer access lines (port hunting groups) and pen recordings of call durations from several terminal lines. The second category of measurements was made by the computer service providers within their computer systems by identifying the arrival and departure times of calls, the amount of central processor time used, the serving port and an identifier of call type. Distributions of call holding time, call interarrival time, CPU usage, and port loading can be obtained from these data. The third category of measurement was the collection of data at computer ports describing the characteristics of such microscopic statistics as intercharacter time. The first two categories of data are being used to formulate traffic and engineering practices. These will be used by telephone company personnel to provide appropriate computer communications by properly configuring existing telephone company equipments.

The third category of data is being employed in the analyses reported here. These data are required to investigate new systems and service characteristics such as the desirability of various transmission speeds or multiplexing methods, as they include detailed information on the timing relationships within a call.

An analytic model of the communications process between a multiaccess computer and a user at a remote console is the vehicle being used to conduct these analyses. The model describes the communications process in terms of random parameters which give the times between characters transmitted through the communications network. All of the parameters are measurable at the communications interface to the computer, i.e., none requires the gathering of data on internal computer processes such as the length of various queues.

The model is used to focus on the user-computer communications process and to exhibit how the characteristics of the computer and of the user affect communications requirements. It is also used to study the converse, i.e., how the constraints of the communications medium affect the user and the computer. The model does not directly represent the detailed characteristics of the computer system or its organization or the internal operation of the user's console. Rather, it reflects the effect of these on the characteristics of the communications signals entering and leaving the computer. From the characteristics of the communications process, however, it is possible to employ the model to predict the effects of changing system characteristics such as improving computer scheduling algorithms or increasing the computer's transmission rate. The following two sections further discuss this model.

The data stream model

The next two sections develop the data stream model, the analytical model used to describe the stochastic interactive communications process between user and computer. In this section, the basic parameters of the model are defined. In the next section, the relationships among the parameters are described and an expression for the holding time of the process is developed where holding time is the duration of a user-computer session.

Figure 1 illustrates the data stream model. A "call" (or a connect-disconnect time period) is represented as the summation of a sequence of time periods during which the user sends characters without receiving, inter-

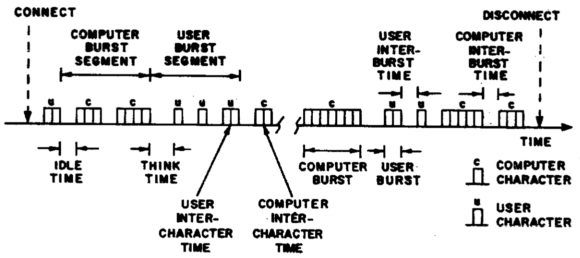


Figure 1—The data stream model

leaved with time periods during which he receives characters without sending. (This implies half-duplex operation. Simple modifications to the model would allow the accommodation of full-duplex operation.) The periods during which the user is sending characters to the computer are defined as user burst segments. The periods during which he is receiving characters sent from the computer are computer burst segments. A user burst segment, by definition, begins at the end of the last character of the previous computer burst segment. Similarly, a computer burst segment begins at the end of the last character sent by the user. The first burst segment of a call begins when the call is established and the last burst segment ends when the call is terminated as measured at the computer interface.

Within a given burst segment, there are periods of line activity and of line inactivity. The first inactive period of a user burst segment is defined as think time. That is, think time is the time that elapses from the end of the previous computer character until the beginning of the first user character in that burst segment. In most cases, think time is employed by the user to finish reading the previous computer output and to "think" about what to do next. The corresponding inactive period in a computer burst segment is called idle time. In some systems idle time represents time during which the user waits for the return of "line feed" after sending "carriage return"; in other systems, idle time represents time during which the user's program is being processed or is in queue. The remaining inactive periods within a burst segment are called intercharacter times and interburst times. A prerequisite for their definition is the definition of a "burst."

Two consecutive characters are defined as belonging to the same burst if the period of inactivity between the characters is less than one-half character width. Thus, each "burst" is the longest string of consecutive characters where the period of inactivity between any two consecutive characters is less than one-half character width. All of the characters in a burst must, of course, be transmitted from the same party (user or computer).

For example, every character of an unbroken string of characters sent at line speed is in the same burst.

For characters within the same user burst, an inactive time between two consecutive characters is called a user intercharacter time. The corresponding parameter for computer bursts is computer intercharacter time. For bursts within the same user (computer) burst segment, the inactive time between two consecutive bursts is called a user (computer)interburst time. Five final parameters of the data stream model are number of user bursts per burst segment, number of computer bursts per burst segment, number of characters per user burst, number of characters per computer burst, and temporal character width (time from start to end of one character).

For a given user-computer environment, a knowledge of the distributions of the parameters defined above allows the calculation of some interesting measures. Examples are distributions for (a) holding time, (b) percent of holding time during which the communication channel carries data, and (c) amount of delay introduced by the computer. The next section shows how some of these distributions can be calculated from the parameters.

Relationships among data stream model parameters

Let the following notation be introduced:

τ	=	holding time of call (seconds)
S	=	number of burst segments in call
T	=	think time (seconds)
I	=	idle time (seconds)
B	=	interburst time (seconds)
N	=	number of bursts per burst segment
M	=	number of characters per burst
W	=	character width (seconds)
C	=	intercharacter time (seconds)

The lower case letters "c" and "u" will be used as superscripts to B, N, M, W, and C to represent "computer" and "user" respectively. For example, N^c will represent the number of computer bursts per computer burst segment. The three indices of summation to be used are:

i —to designate the ith burst segment,

j —to designate the jth burst of a given burst segment,

and

k—to designate the kth character of a given burst.

In summing expressions over these indices, the primary index will be shown as a subscript and the secondary index (or indices), if any, will be enclosed in parentheses.

Using this notation, it is possible to construct an equation relating the holding time of a call to its component parts in the following manner:

a. In burst segment 2i, in the jth burst, the amount of time required by the kth user character is $W^u_k(2i, j)$. Summing over all k such characters in the burst, the time required is

$$\sum_{k=1}^{M^u_j(2i)} W^u_k(2i, j).$$

Summing over all bursts in the burst segment gives

$$\sum_{j=1}^{N^u_{2i}} \sum_{k=1}^{M^u_j(2i)} W^u_k(2i, j) .$$

If one defines all burst segments where i is even as user burst segments and assumes that the number of burst segments per call (S) is always even, the total contribution of user character times to total holding time is

$$\sum_{i=1}^{S/2} \sum_{j=1}^{N^u_{2i}} \sum_{k=1}^{M^u_j(2i)} W^u_k(2i, j) .$$

Since S is usually large the error introduced by assuming S even, even when it is not, is small. Assuming the odd numbered burst segments are computer burst segments, the corresponding contribution of computer character times is

$$\sum_{i=1}^{S/2} \sum_{j=1}^{N^c_{2i-1}} \sum_{k=1}^{M^c_j(2i-1)} W^c_k(2i - 1, j) .$$

b. A corresponding argument shows the total contributions of user intercharacter times are

$$\sum_{i=1}^{S/2} \sum_{j=1}^{N^u_{2i}} \sum_{k=1}^{[M^u_j(2i) - 1]} C^u_k(2i, j)$$

and of computer intercharacter times are

$$\sum_{i=1}^{S/2} \sum_{j=1}^{N^c_{2i-1}} \sum_{k=1}^{[M^c_j(2i - 1) - 1]} C^c_k(2i - 1, j) .$$

c. The total amount of user interburst time is

$$\sum_{i=1}^{S/2} \sum_{j=1}^{[N^u_{2i} - 1]} B^u_j(2i) ,$$

and the total computer interburst time is

$$\sum_{i=1}^{S/2} \sum_{j=1}^{[N^c_{2i-1} - 1]} B^c_j(2i - 1) .$$

d. Total think time is

$$\sum_{i=1}^{S/2} T_{2i} ,$$

and total idle time is

$$\sum_{i=1}^{S/2} I_{2i-1} .$$

The sum of these components is the holding time for a call. That is, the time of each burst segment summed over all burst segments is the holding time. The time of a burst segment equals the sum of the durations of all bursts, interburst times, and the think (or idle) time in that burst segment. The duration of a burst is equal to the sum of the character times and the intercharacter times contained therein. That is, the holding time of call ℓ, τ_ℓ, is

$$\begin{aligned}\tau_\ell = \sum_{i=1}^{S/2} \Bigg\{ \Bigg[T_{2i} &+ \sum_{j=1}^{[N^u_{2i} - 1]} B^u_j(2i) \\ &+ \sum_{j=1}^{N^u_{2i}} \sum_{k=1}^{M^u_j(2i)} W^u_k(2i, j) \\ &+ \sum_{j=1}^{N^u_{2i}} \sum_{k=1}^{[M^u_j(2i) - 1]} C^u_k(2i, j) \Bigg] \\ &+ \Bigg[I_{2i-1} + \sum_{j=1}^{[N^c_{2i-1} - 1]} B^c_j(2i - 1) \\ &+ \sum_{j=1}^{N^c_{2i-1}} \sum_{k=1}^{M^c_j(2i - 1)} W^c_k(2i - 1, j) \\ &+ \sum_{j=1}^{N^c_{2i-1}} \sum_{k=1}^{[M^c_j(2i - 1) - 1]} C^c_k(2i - 1, j) \Bigg] \Bigg\} . \quad (1)\end{aligned}$$

Knowing the distributions for the 12 parameters in Equation (1), it is theoretically possible to solve directly for the distribution of holding time. The mechanics of finding the solution are prohibitive, however, except for very restricted cases. One method of solving (1) is to find the moments of holding time rather than the complete distribution. This approach will be used here and, in fact, it will be sufficient for our purposes to solve merely for the mean value of holding time. In order to arrive at the solution, we assume that the random variables are stationary and mutually independent.*

Taking the expected value of both sides of (1), we obtain

$$\begin{aligned}\tau = (S/2)[T &+ B^u(N^u - 1) + N^uM^uW^u \\ &+ N^uC^u(M^u - 1) + I + B^c(N^c - 1) \\ &+ N^cM^cW^c + N^cC^c(M^c - 1)] \quad (2)\end{aligned}$$

where the symbol for each variable without a subscript implies its mean value. For further analysis, Equation (2) may be separated into four parts each having its own functional significance:

a. user send time (the total amount of time during which user characters are being transmitted)

$$= (S/2)(N^uM^uW^u) ,$$

b. computer send time (the total amount of time during which computer characters are being transmitted)

$$= (S/2)(N^cM^cW^c) ,$$

c. user delay (the sum of all inactive periods during user burst segments)

$$= (S/2)[T + B^u(N^u - 1) + N^uC^u(M^u - 1)] ,$$

d. computer delay (the sum of all inactive periods during computer burst segments)

$$= (S/2)[I + B^c(N^c - 1) + N^cC^c(M^c - 1)] .$$

The sensitivity analysis performed in the next section is an investigation of the properties of these four parts and includes a discussion of how their interrelation affects holding time. If any of these parts can be

* Analyses of these assumptions have exposed their limitations. However, these assumptions have been shown to be reasonable for the analyses and conclusions of this paper.

reduced without increasing others, holding time can be reduced leading to possible cost savings.

Before discussing such analyses, it may be well to indicate what values have been observed for each of the 12 parameters. This will allow us to concentrate our attention on those parameters and those measures of parameters that promise to be the areas of greatest possible holding time reduction.

Collected data and sensitivity analyses

During the current study, data have been gathered on a large number of calls to each of several multiaccess computer systems. For each system, the data have been partitioned into sets representing each of the 11 random parameters (the twelfth parameter, character width, is a constant). Probability density functions have been fitted to the data collected on each parameter from each system.[5]

Data from three of the systems are discussed in this paper. These systems are labeled A, B, and C. Systems A and B have the same computer equipment and basically the same mix of user applications (programming—scientific). System C has computer equipment different from that of the other two systems and its mix of user applications is primarily business oriented. All three systems serve low-speed teletypewriter-like terminals. System B is rather heavily "loaded" compared to Systems A and C. Table II summarizes these characteristics for Systems A, B, and C.

Table II—Characteristics of systems studied

	System A	*System B*	*System C*
Computer Type	Brand X	Brand X	Brand Y
Transmission Speed (Characters/sec)	10	10	15
Primary Application	Scientific	Scientific	Business
Load	Moderate	Heavy	Moderate

Table III summarizes the measured values of the model parameters. To ensure the privacy of the three systems under discussion, these values are not shown on a per system basis. Rather, for each parameter,* an average value $\hat{\mu}$ is given where $\hat{\mu}$ is the average of the three system averages for the given parameter. The numbers in the column headed $\sigma_{\hat{\mu}}$ are the standard deviations of the three numbers averaged in $\hat{\mu}$ for each parameter. The analyses subsequently reported, however, are based on the actual per system average values of these parameters.

* As the character widths W^u and W^c are treated in the model as random variables, they are included in Table III for completeness. In the three systems discussed, however, they were constant as can be derived from Table II.

Table III—Average parameter values

		$\hat{\mu}$	$\sigma_{\hat{\mu}}$
S	—No. of Burst Segments	82.	37.
T	—Think Time (sec.)	4.3	3.4
I	—Idle Time (sec.)	.65	.48
B^u	—User Interburst Time (sec.)	1.6	.90
B^c	—Computer Interburst Time (sec.)	16.	25.
N^u	—No. of Bursts/User Burst Seg.	11.	3.1
N^c	—No. of Bursts/ Computer Burst Seg.	3.3	2.8
M^u	—No. of Characters/ User Burst	1.1	.12
M^c	—No. of Characters/ Computer Burst	47.	27.
$W^u(=W^c)$	—Character Width (sec.)	.089	
C^u	—User Intercharacter Time (sec.)	.00021	.00023
C^c	—Computer Intercharacter Time (sec.)	.00030	.000090

One characteristic of the data summarized in Table III deserves further comment. It is that measures which should be most sensitive to computer characteristics seem to just that. For example, the users of both Systems A and B have predominantly programming-scientific applications and the average numbers of characters per user burst segment (N^uM^u) are 9.2 and 10.7 for Systems A and B, respectively, versus 13.8 for the primarily business oriented users of System C. Such relationships prevail in spite of widely different average computer delays. The average amount of time spent per computer burst segment in interburst delay, $(N^c\text{-}1)B^c$, is 1.4 seconds in System A and 35.8 seconds in System B.

Table IV—Mean values of holding time components

	System A	*System B*	*System C*
Average Holding Time, τ			
Minutes	17.	34.	21.
Average User Send Time, $(S/2)(N^uM^uW^u)$			
Minutes	0.50	0.45	0.96
% of τ	3%	1%	5%
Average Computer Send Time, $(S/2)(N^cM^cW^c)$			
Minutes	5.7	4.5	7.5
% of τ	33%	13%	35%
Average User Delay, $(S/2)(T + B^u[N^u - 1] + N^uC^u[M^u - 1])$			
Minutes	10.	12.	11.
% of τ	58%	35%	53%
Average Computer Delay, $(S/2)(I + B^c[N^c - 1] + N^cC^c[M^c - 1])$			
Minutes	0.95	17.	1.5
% of τ	6%	51%	7%

Table IV summarizes the macroscopic characteristics of these data as they contribute to holding time. An inspection of the table leads to the following observations:

Observation 1: The average holding time for the heavily loaded system (System B) is considerably larger than for the lightly loaded Systems A and C (94 percent and 60 percent larger).

Observation 2: For the lightly loaded systems, A and C, Computer Delay is less than 10 percent of the total holding time but for System B Computer Delay accounts for over half of the total holding time.

Observation 3: User Delay is a significant component of holding time in all three systems, and in each case, is between 10 and 12 minutes.

Observation 4: User Send Time is less than 5 percent of total holding time in each system and is not a significant contributor to holding time.

Observation 5: Computer Send Time is smallest in both absolute value and in percent holding time for the heavily loaded System B.

These five observations lead to three broad areas of interest that are discussed in the next three sections. The first area is the relationships between holding time and computer delay (Observations 1 and 2). The second is the relationships between holding time and user characteristics (Observations 3 and 4). The third is the relationship between holding time and computer send time (Observation 5).

Relationships between holding time and computer delays

The Computer Delay times shown in Table IV indicate a large variability among computer systems. This section investigates this variability.

There are a number of convenient measures that can be used to describe "computer load" and "computer delays." The "load" on a computer is a function of the number of simultaneous users who are "active" (in queue waiting for the computer to run their program or output to them), and the characteristics of user programs. For the purposes of the present discussion, data availability requires the use of "simultaneous users" as our measure of computer load. The manner in which a computer system reacts to a fluctuating load is a function of many additional variables, including characteristics of the scheduler.

Average computer delay may be calculated as the average total amount of computer delay per call, i.e., the sum of all the idle times, computer interburst times, and computer intercharacter times in a call.* This method was used in Table IV to demonstrate the effects of total computer delay on the average holding times of the three computer systems. It is also beneficial to examine the individual components of total computer delay. For example, it appears reasonable to divorce our measure of computer delay from the number of burst segments in a call by considering computer delay per burst segment. This new measure is reasonable because the number of burst segments

* Symbolically, average total computer delay per call is $(S/2)(I + B^c[N^c - 1] + N^cC^c[M^c - 1])$.

Table V—Components of average computer delay per computer burst segment (all times in seconds)

	System A	*System B*	*System C*
Average Computer Delay per Computer Burst Segment (Δ)	1.7	37.	1.4
Average Idle Time per Computer Burst Segment	.33	1.2	.41
% of Δ	19%	3%	28%
Average Interburst Time per Computer Burst Segment	1.4	36.	.99
% of Δ	79%	97%	69%
Average Intercharacter Time per Computer Burst Segment	.03	.02	.04
% of Δ	2%	—	3%

per call appears to be highly sensitive to user application type. Thus, calculating computer delay per burst segment reduces the dependence of our results on user application. We now consider contributions from average idle time (I), from average intercharacter times ($N^cC^c[M^c-1]$), and from average interburst times ($B^c[N^c-1]$). These three components of average computer delay per computer burst segment are shown in Table V.

As can be observed, the delays introduced by interburst times are the majority of all computer delay components in each system. The explanation for the relative sizes of these three components is as follows:

a. The characteristics of user programs are such that two or more quanta of execution time are required for a run to completion but output is generated by each quantum; and
b. The combination of system load, output buffer size, and characteristics of the scheduler preclude the immediate availability of additional output when the transmission of a computer burst is completed.

Because average *I*nter*B*urst *T*ime is the largest single contributor to average computer delay, it will be denoted by a special symbol, I-B-T. Figure 2 shows the relationships between holding time and I-B-T for each of the three computer systems. The three points in Figure 2 associated with computer systems are the observed values of holding time and I-B-T for those systems. The three lines are generated by changing I-B-T for each of the systems while holding all other parameters fixed. The slope of each line is equal to one-half the number of burst segments per call, i.e.,

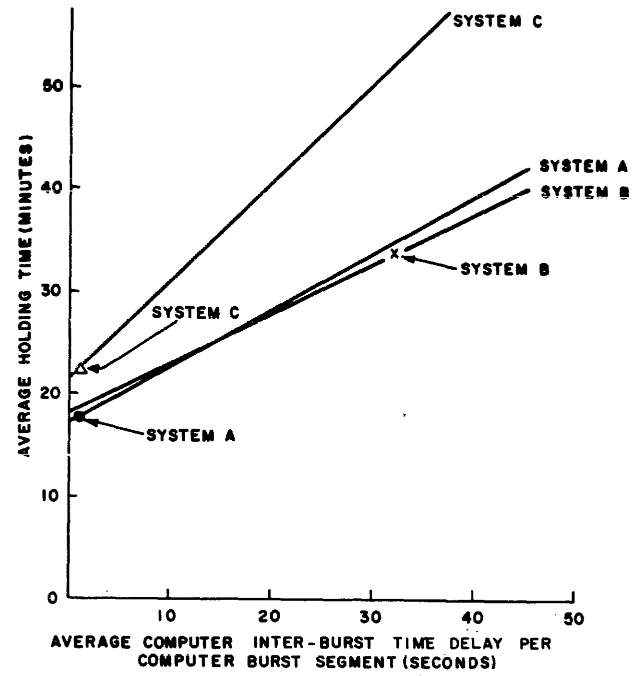


Figure 2—Average holding time versus average computer interburst time delay per computer burst segment

S/2, because when every other factor in Equation (2) is held constant, it becomes

$$\text{Holding Time} = \text{Constant} + (S/2)\ (\text{I-B-T}). \qquad (3)$$

Note that the lines for Systems A and B are close together. As these two systems have similar configurations of hardware and software, support is given to the conjecture that increasing the load on System A would lead to values of I-B-T and holding time comparable to those of System B. Conversely, deloading System B should result in these parameters having values comparable to those of System A.

Next, as it appears that holding time is a function of I-B-T and I-B-T is in turn a function of the loading on the computer and hence on the number of simultaneous users, an expression relating I-B-T and number of active users can be established.

To establish the relationship between I-B-T and number of simultaneous users, both quantities were measured on the systems as a function of time of day. The solid curve in Figure 3 indicates the average number of simultaneous users of System A for 15-minute periods of the day. The average I-B-T's were calculated for hourly periods on data from System A and a least squares fit of a variety of curve types was investigated. For these data the best fit is

$$\text{I-B-T} = (0.18) \exp (0.13\ u) \qquad (4)$$

where

$$u = \text{number of simultaneous users,}$$

or

$$u = (7.7)\ (1.7 + \ln \text{I-B-T}). \qquad (5)$$

The dashed curve in Figure 3 is a plot of u versus time of day by using Equation (4) and the actual measurements of I-B-T versus time of day. This fit seems to reflect the major characteristics of the data as shown by the solid line in the sense that at least the morning and afternoon busy periods are reflected along with the intervening noontime lull.

Using (4), a plot can also be made relating I-B-T and the number of simultaneous users. Figure 4 shows this relationship as well as three curves showing the effects of I-B-T on holding times in Systems A, B, and C. These latter curves are plotted from Equation (3) after substitution of (4). Figure 4 indicates that above some threshold (represented by the knees of the curves) the computer's grade of service deteriorates rapidly as additional users are accepted.*

Relationships between holding time and user characteristics

Tables VI and VII, below, are used to illustrate several relationships between holding time and user characteristics in this section and between holding time and computer send times in the next section.

* Here an analogy can be drawn between the manner in which a multiaccess computer reacts to increasing loads and the fashion in which a telephone switching system reacts to overloads. As the link occupancy increases in a telephone office, the probability that a path through the switching networks cannot be found for an incoming call increases in a manner similar to the computer delay curve in Figure 4 (Reference 6). Such failures to complete connections can cause the common control equipment to generate additional attempts that consume additional real time. In computer systems, the analogous work is the "swapping" of user programs into and out of central processor core. For a further discussion of this computer problem see Scherr,[1] Raynaud,[2] Greenberger,[7] and Coffman.[8]

Table VI—Send time information
(all times in minutes)

	System A	*System B*	*System C*
Average Holding Time (τ)	17.	34.	21.
Average Total Send Time (R) = $(S/2)(N^uM^uW^u + N^cM^cW^c)$	6.2	5.0	8.4
% of τ	36%	15%	40%
Average User Send Time $(S/2)(N^uM^uW^u)$	.50	.45	.96
% of τ	3%	1%	5%
% of R	8%	9%	11%
Average Computer Send Time $(S/2)(N^cM^cW^c)$	5.7	4.5	7.5
% of τ	33%	13%	35%
% of R	92%	91%	89%

Table VI shows, for each system, the average holding time (τ), the average total send time** (R), and the average user and computer send times. These quantities are measured both in minutres and, for the latter three categories, as a percentage of holding time.

Table VII is constructed identically for delay quantities rather than for send time quantities.

** Average total send time is the sum of average user send time and average computer send time.

Table VII—Delay information
(all times in minutes)

	System A	*System B*	*System C*
Average Holding Time (τ)	17.	34.	21.
Average Total Delay (D)	11.	29.	13.
% of τ	64%	85%	60%
Average Total User Delay	10.	12.	11.
% of τ	58%	35%	53%
% of D	92%	40%	88%
Average Total Computer Delay	.95	17.	1.5
% of τ	6%	51%	7%
% of D	8%	60%	12%

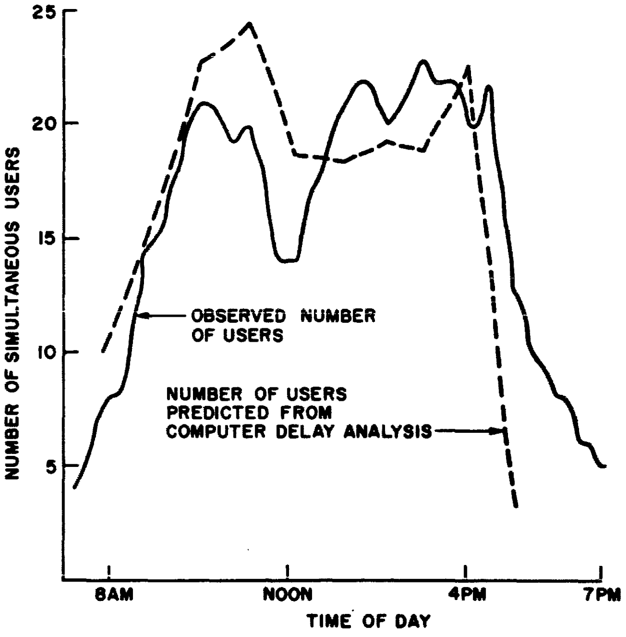


Figure 3—Number of simultaneous users in system A versus time of day

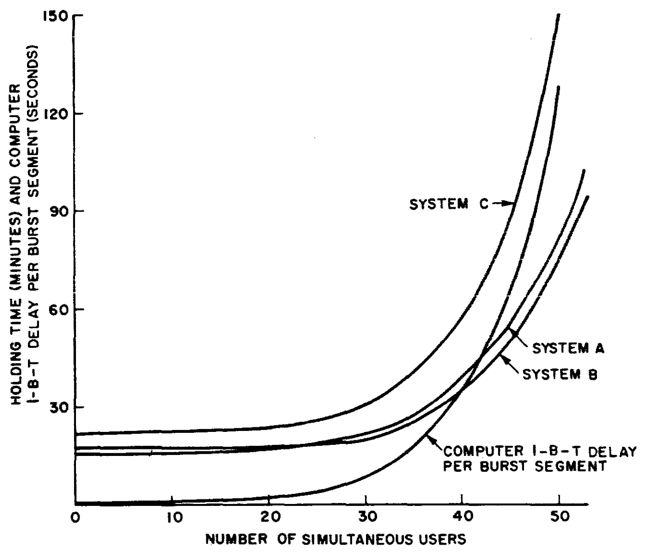


Figure 4—Average holding times and computer interburst time (I-B-T) delay per burst segment versus number of simultaneous users

Table VI indicates that (a) in all three systems average user send time accounts for less than five percent of average holding time and less than 12 percent of average total send time, and (b) the users of System C inputted about three times† as many characters as the users of the other two systems. A conclusion that can be drawn from (a) is that user send time is an insignificant contributor to holding time. Even if average user send time increased by a factor of three, the increase in holding time would be only one to two minutes, assuming that total user delay remains fixed. The reasons for (b) are probably a combination of, first, the business oriented applications of many of the system's users and, second, the rather low computer delays experienced in System C. It is possible that this increase in user input volume in System C is encouraged by the small computer delays experienced in that system. The same factor could also be partially responsible for the greater degree of on-line interaction in System C which when compared to Systems A and B had about double the average number of burst segments per call.

Table VII indicates that average total delay D (the sum of average user and computer delays) accounts for more than half of average holding time. The lowest percentage is for System C where average total delay is

† Recall that terminals in System C operated at 15 characters per second versus 10 characters per second in the other two systems.

60 percent of average holding time; the highest is System B with 85 percent. Of these delays, users contributed from 40 percent (System B) to 92 percent System A). Another observation is that the absolute value of average user delay is between 10 and 12 minutes in all three systems. This consistency is rather remarkable when one considers the diversity of other parameters that affect user delay.

One might conjecture further about how user send time and delay characteristics would be affected by different transmission speeds, different program applications, different levels of user sophistication, and many other variables. For example, the user who inputs his prepared program from punched tape eliminates the user delay introduced by the user who performs this function by the hunt and peck method. An analogous "user" characteristic is computer send time which is determined almost entirely by the amount of computer output requested by the user. The next section discusses how this measure affects holding time.

Relationships between holding time and computer send time

Table VI shows that the system with the highest load, i.e., System B, has the smallest user send times and computer send times in both absolute value and in percent of holding time. This fact may be partially caused by the user's tendency to limit the amount of on-line I/O when he experiences long computer delays. The second, and more useful, observation that can be made from Table VI is that in systems which are not heavily loaded, average total send time may be on the order of 35 percent to 40 percent of average total holding time. Of this time, approximately 90 percent is computer send time.

We may infer from these data that, barring changes in user patterns and other influencing factors, holding time may be materially reduced by providing a high-speed channel from the computer to the user and a high-speed printer, or other display device at the user's location. This system redesign would enable a decrease in W^c, computer character width, with a corresponding decrease in

$$\text{Holding Time} = \text{Constant} + (S/2)\,(N^cM^cW^c). \quad (6)$$

Figure 5 is a plot of average holding times versus computer-to-user channel speed assuming all other factors remain constant. Of the three systems, two of them, viz., Systems A and B, have computer channel speeds of 10 characters per second. System C transmits at 15 characters per second. Note that if System C transmitted at 10 characters per second, its average holding time would be expected to increase to about 25 minutes from 21 minutes.

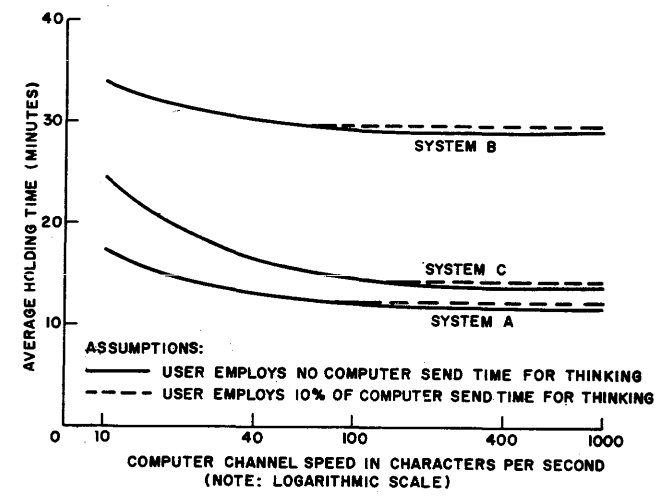


Figure 5—Average holding time versus computer transmission rate for three long holding time systems

If computer channel speed were infinite, the average holding times for Systems A, B, and C would be 11.7, 29.3, and 13.7 minutes, respectively. To approach these minima within ten percent would require computer channel speeds of about 60 ch/sec for System A, 20 ch/sec for System B and 100 ch/sec for System C. If a channel speed of 360 char/sec were available, the computer send times would be less than one half minute and less than 2.5 percent of average holding time in each system.

At this point one might conjecture that there are at least two components of holding time that are likely to increase if computer transmission rates are increased. The first is think time because, in at least some instances, the user utilizes computer send time to read the output he receives. Hence, if the computer outputs the same number of characters in a much shorter time interval, the user may increase his think time in order to do the same amount of reading and thinking. This interplay of responses and transmission rates has effects on holding time in that it suggests the existence of some upper bound on computer transmission rate beyond which decreases in computer send time are matched by equal increases in user think time. The result is that average holding time cannot be further reduced by increasing computer transmission rates. In order to attach quantitative significance to this conjecture let us assume that the average user employs currently 10 percent of computer send time for reading and thinking.* This

* Ten percent is certainly a high figure for the user who is listing his program to provide a paper copy for filing purposes but is low for the user who is checking every comma and parenthesis in order to find a program bug.

assumption implies that a tenfold increase in computer transmission rate will result in minimum average holding time. For Systems A and B this rate is 100 characters per second (1000 words per minute) and yields holding time of about 12.5 minutes and 29.5 minutes, respectively. These average holding times are 28 percent and 13 percent less than the current holding times and seven percent and one percent greater than the theoretical minima shown in Figure 5 for Systems A and B. For System C, the corresponding transmission rate is 150 characters per second. This would reduce the average holding time in System C by 31 percent to about 14.5 minutes which is six percent greater than the theoretical minimum of 13.7 minutes. The minimum holding times implied by this assumption are shown as dashed lines in Figure 5.

The second variable that may naturally increase if computer transmission rates increase is the quantity of output requested by the user. For example, if the computer transmitted at 360 characters per second, computer send time would be less than 2.5 percent of average holding time in all three systems, and users may find it quite convenient to request two or three times as much output as they do presently. This increase in output will not severely affect average holding times, however, as even a tripling of computer send time would increase average holding time by less than a minute assuming a 360 character per second computer rate.

It should be noted that these figures are heavily dependent on such factors as user application. For example, if the application is not scientific programming but rather inquiry-response one might anticipate drastically shorter holding times. Consider the telephone directory assistance operator who makes a five second query of a computer that responds in one second with the beginning of a 1000 character transmission to be displayed with a video terminal. Assume that "holding time" is defined to be

$$\tau = \text{operator keying time} + \text{computer response time} + \text{computer transmission time}$$

or

$$\tau = K + I + 1000\ W_c\,.$$

For the numbers we have assumed,

$$\tau = 6 \text{ seconds} + 1000\ W_c\,,$$

his program to provide a paper copy for filing purposes but is low for the user who is checking every comma and parenthesis in order to find a program bug.

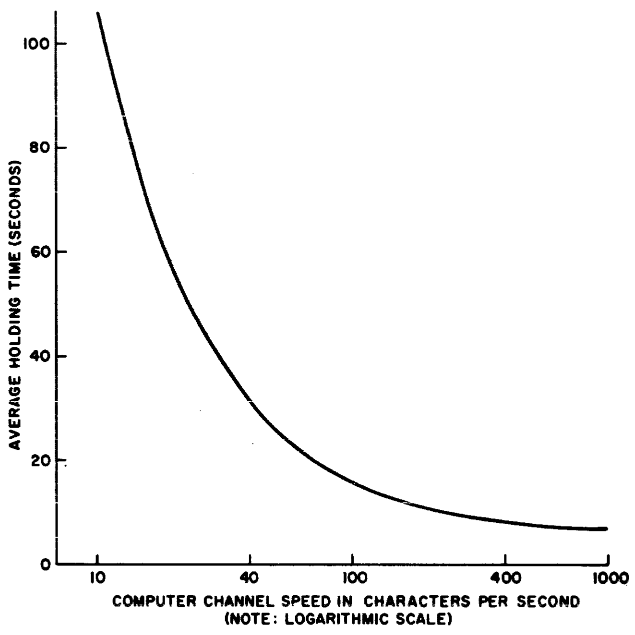


Figure 6—Average holding time versus computer transmission rate for hypothetical short holding time system

and at 10 characters per second, this results in a 106 second holding time of which 94 percent is computer send time. At 360 characters/second, τ = 8.8 seconds but computer send time is still 32 percent of τ. At 4000 characters/second, τ = 6.25 seconds of which four percent is computer send time. Figure 6 shows a graph of holding time versus computer channel speed for the short holding time example we have assumed.

SUMMARY AND CONCLUSIONS

The results of a study of the communications considerations of serving multiaccess computer systems have been presented. A model of user-computer interaction, as observed at the communications interface, has been developed. Summary data from three "long holding time" computer systems have been given for the parameters of this model.

Examination of these data has revealed that

a. Computer introduced delays can be a large component of holding time and, above some threshold, are acutely sensitive to the number of simultaneous users. The largest component of computer delay occurs during those periods when the computer is outputting to remote users. The conclusions to be drawn from these findings stem from the consideration of holding time as being composed of periods of computer outputting activity, and periods of no computer outputting.

Of the inactive periods, some time is due to user-dependent delays and some to computer-dependent delays, some, such as execution time, may not be reducible and others, such as delays due to overhead, can be reduced. It should be noted that changes in the computer system such as changes to the scheduling algorithm,[7,8,9] or changes to the communications control unit[10] can strongly influence computer delays. Thus, it is within the *computing* system that some reductions in holding time may be made resulting in communications economies. As not all computer delays can be eliminated in a heavily loaded system, the technical and economic feasibility of employing data multiplexers at the computer to decrease the number of access lines should be explored.

b. The average number of characters sent by the computer to the user is an order of magnitude greater than the number of characters sent by the user to the computer. If other parameters did not change drastically, the availability of higher transmission rates for computer outputting would effect significant reductions in average holding time. A computer transmission rate of 360 characters per second would reduce total computer send time below one half minute per call. To achieve this rate requires either high speed or asymmetric data sets and correspondingly higher speed output terminals.

c. Delays introduced by the user are a significant contributor to average holding time. The average user delays in the three systems reported are remarkably close in absolute values. As user delays are appreciable, the multiplexing of inputs from user terminals which are geographically clustered appears attractive and is being studied.

The data analyzed in this report are from systems having primarily scientific and business problem-solving applications. The users of such systems demonstrate a wide range of sophistication in their use of these systems. As users educate themselves in the efficient use of multiaccess computers and their terminals, the data traffic characteristics of these systems will change.

Studies of multiaccess computer communications systems are continuing. Data are being collected from systems with markedly different terminal types, average holding times, and user applications. Analyses of these data will allow the characterization of such systems in a manner analogous to that reported above for "long holding time" systems.

Implications of results

The implications of the results of this study extend into several aspects of computer communications. The study has produced quantitative indications of the degree to which computer operations can influence such communications parameters as holding time. The developers of computer systems, in turn, have noted that the provision of computer hardware and software to accommodate data communications is a major problem area.[11],[12] In order to jointly optimize the computation-communication solution to the problem, it is apparent that closer coordination between the computer and communications systems designers would be extremely fruitful in terms of economic and technological improvements to overall systems design.

One of the impediments to finding rapid and robust solutions to the problems of multiaccess computer communications has been the unavailability of data descriptive of the user-computer interaction process. The acquistion of the data reported here is a contribution toward the removal of that obstacle.

These data are currently being used in systems engineering studies at Bell Telephone Laboratories to further define the systems requirements for new systems and services to satisfy the needs of the multiaccess computer community.

The analyses made of these data support for the first time in more than qualitative terms some proposals proffered in the past as solutions to the data communications problems associated with multiaccess computer systems. The delays which are introduced by both user and computer suggest the possibilities for effective employment of multiplexing techniques. For example, it has been shown for the systems studied that the average total send time (the sum of user send time and computer send time) is as little as five to nine minutes. This corresponds to 15 to 40 percent of average holding time. For the other 60 to 85 percent of average holding time the communications channel is idle. One method of obtaining higher utilization of these facilities is by time division multiplexing.* The following assumes a multiplexing technique in which the user channel is independent of the computer channel. Only one to five percent of average holding time (or three to eight percent of the average user burst segment) is user send time. Thus, for 95 to 99 percent of an average call, the user-to-computer channel is idle and could be made

* Multiplexing is not always economical, of course, despite the large idle times. Other important considerationss involve the geographical placement of terminals and computers and several statistical traffic characteristics other than average occupancy.

available to additional users. Average computer send times for the three systems were from 13 to 35 percent of average holding time (21 to 86 percent of the average computer burst segment) indicating that higher usage of the computer-to-user channel may be realized by the use of appropriate multiplexing techniques.

The asymmetric nature of the data flow in multi-access computer systems suggests that different transmission treatments may be appropirate for computer-to-user versus user-to-computer transmissions. The large volumes of computer-to-user data are an order of magnitude greater than volumes in the opposite direction. The provision of computer transmission rates of 100 to 200 characters per second could reduce average holding times up to 30 percent. As the user is capable of generating characters for transmission at a much slower rate, the application of asymmetric channels or data sets receives quantitative support and is now being studied. Provision for higher computer transmission rates would require, of course, user terminals with accordingly higher input rates.

The final conjecture receiving quantitative support from the above analyses is that users themselves contribute substantially to the communications costs of their real-time computer access calls by introducing delays. Some of these delays are likely to decrease as users gain proficiency. Others are due to the inveterate characteristics of human users. As they pertain to the use of communications and to the use of computers[1,3] these characteristics are being intensively studied to enable the design of versatile and responsive computer/communications systems.

ACKNOWLEDGMENTS

Many people have contributed their efforts to various parts of this study. Data acquisition was accomplished with the considerable help of the American Telephone and Telegraph Company and the Bell System Operating Companies. Contributions to the model and the analyses and many helpful criticisms were made by Messrs. E. Fuchs, R. J. Price and R. J. Roddy, all of Bell Telephone Laboratories.

Our special thanks are extended to the companies whose computer systems are being studied. Without their full permission and very helpful cooperation these analyses would not be feasible.

REFERENCES

1 A L SCHERR
An analysis of time-shared computer systems
Massachusetts Institute of Technology Project MAC
Thesis MAC-TR-18 June 1965

2 T G RAYNAUD
Operational analysis of a computation center
Operations Research Center MIT July 1967

3 H SACKMAN
Experimental investigation of user performance in time shared computer systems retrospect, prospect, and the public interest
System Development Corporation May 1967

4 G E BRYAN
JOSS: 20,000 hours at the console, a statistical summary
Rand Corporation August 1967

5 P E JACKSON
A fourier series tes' of goodness of fit
To be published

6 ANON.
Switching systems
AT & TCo 1961

7 M GREENBERGER
The priority problem
MIT November 1965

8 E G COFFMAN JR
Analysis of two time sharing algorithms designed for limiting swapping
J A C M July 1968

9 E G COFFMAN JR L KLEINROCK
Computer scheduling methods and their countermeasures
Proc S J C C 1968

10 L J COHEN
Theory of the operating system
The Institute for Automation Research Inc 1968

11 W F BAUER R H HILL
Economics of time shared computing systems—Part II
Datamation Vol 13 No 12 41–49 December 1967

12 W E SIMONSON
Data communications: The boiling pot
Datamation Vol 13 No 4 22–25 April 1967

13 W W CHU
A study of the technique of asynchronous time division multiplexing for time-sharing computer communications
Proc of the 2nd Hawaii International Conference on Systems Sciences January 1969

A communications environment emulator

by JACK M. PEARLMAN and RICHARD SNYDER

Honeywell, Inc.
Waltham, Massachusetts

and

RICHARD CAPLAN

Consultant
New York, New York

The problem

Communications based systems and software, especially complex systems and software, present an exceptionally difficult checkout and debug problem to the implementor. Conventional techniques require him to check his often times labyrinthine maze of response paths in a plodding, single step by single step, long drawn out and expensive manner. He checks the non-communications or non-real time portions of his program easily enough. The methods of testing standard peripheral functions are well enough known and adequate tools exist for the job. The techniques for checking basic data processing logic are available with every batch oriented system. Communication based system checkout is complicated by the real time nature of the systems. The give and take of on-line terminals, the multiplicity of choice, the need to process data in various stages of disarray, the necessity of leaving some partially complete processing sequence and returning to it at a later time, these are the problem factors. These are the things which make conventional checkout procedures a millstone around the neck of a system builder.

He is required to generate huge quantities of test data, each item of which is keyed to a specific path, expecting a specific response. He simulates as well as he can the action of his operational system with a console or perhaps with a card reader or maybe a single actual terminal. The usual scenario then calls for him to use several actual terminals and operators, with the attendant confusion, logistic difficulties and expense. The final step is a full blown system test sequence using all the equipment and all the operators and he hopes that all the parts are exercised and that all the problems are solved.

This technique works, to a point. It does produce a workable system. It does, eventually, succeed in resolving a large percentage of possible problems. The cost of doing so, however, is extremely high. Devising a specific datum for a specific path test is time consuming, especially since the number of paths may be counted in the hundreds or thousands. Using live terminals and live operators presents a major organizational problem and requires large cash outlays. Such a system too, violates a prime rule of scientific investigation—reproducibility of an experiment. The programmer is rarely positive that a fix works since he can never exactly recreate the conditions causing failure.

The solution

Taking these objections, and others, together, we decided to find a better way to check and debug communications based systems and system software. The result of our efforts, is "The Honeywell Communications Environment Emulator" (HCEE).

HCEE is a multi-purpose communications network simulator whose prime purpose is to aid in the checkout and debugging of communication software, real time or otherwise. A secondary purpose is to permit experimentation with software and remote terminal configurations where the terminal itself is non-existent. A tertiary reason for being is to evaluate communications software by determining its operational limits and its response to unusual stress.

The system may be quantitatively defined in terms of its design goals. It will simulate up to 63 lines, with several distinctly different classes of terminal represented, up to eight terminals per line and generate at least 70,000 100-character messages per hour on an H-1200 computer. It will require a minimum 65K C.P. with about 30K taken up by instructions. It will handle as many message types as a user cares to define and check for as many response types as he wishes. The system will generate as many message variations as a user provides defining entries for, in some cases using a random selection process, in others using a round-robin technique. HCEE will eventually be able to introduce perturbations into a data stream to simulate line control errors and other line associated faults.

The Emulator will reside in its own processor, with the system under test (target) in second C.P. The two computers communicate via standard communications hardware. The target system thinks that it sees a real communication network, with real terminals and operators, in the outside world. Actually HCEE generates all queries and responds to all answers by simulating or emulating the characteristics of the actual terminals and their operators. This approach to checkout is itself complex and by no means inexpensive. But it is far less complex and far less expensive than conventional checkout means and carries with it many distinct advantages. Use of a computer permits reproducibility and creation of exhaustive system evaluation reports. By recording every message transmitted (control and data) and every message received, along with the system time, it becomes possible to "play back" an entire sequence as a way of validating changes to the target system. These same records become the basis for a complete series of reports describing the operation of the system in detail and the success with which various classes of messages were handled, including deliberately created error messages. The computer, too, permits automatic message generation and response analysis based on skeletons defined by the user.

The product

The form which the HCEE design took was partially determined by a number of criteria for use, not the quantitative ones described earlier, but qualitative ones. One major criterion was that the system be modular. By modularity we imply that the component elements be so loosely related to one another (or decoupled) that any element could be readily replaced by a similar component of greater or less complexity. That is, except for some minimum capability, new routines could be added and/or existing ones deleted from the system. This criterion is essential to a system such as HCEE since its utility will be partially dependent upon our ability to adapt it to handle changed operating conditions, and provide capabilities not originally considered. Modularity is achieved by use of elaborate table structures and list processing techniques which insulate processing routines from one another. All contact between routines is through tables, lists or well defined interfaces to small central control programs.

A second important criterion was that the system be able to generate large message volumes within user defined response constraints. This implies that the time limitations associated with program or data storage on external media probably would become intolerable. Consequently it was decided that the entire program and its data base be core resident.

A further goal was to make the program usable internally by Honeywell as a systems checkout tool and eventually externally by our customers, with a third potential user being independent service bureaus. These were some of the factors which led us toward residence in a separate C.P., with independent communications capability. We could now use this tool to debug a target system hundreds of miles away and work with target systems residing in non-Honeywell computers.

The last major criterion considered required that the generated system be independent of any existing operating system. This decision was made to insure usability in a smaller computer than would otherwise be necessary and to insure better control over HCEE operation by not being tied to operating system conventions. We will use operating system MOD 2 to generate the system however, in order to take advantage of the macro assembly and linkage editing and loading facilities of MOD 2.

Phase structure

Functionally, HCEE may be considered as having four discrete phases; Test Generation, Test Initialization, Test Execution and Test Result Analysis.

During Test Generation, the data and environment Specifications which define the test case are integrated into the HCEE execution phase structure. This operation is accomplished through application of the Macro Assembly facility. Macros were selected as the prime configuration medium for several reasons: the facility exists; it is fairly powerful, it is well understood; macros are relatively easy to use; they require minimal manpower to implement; they are, of all alternatives, the cheapest to use. Macros were an expedient choice but more importantly, they do not impose any major limitation upon either the design or the final use of the system. One further consideration is the relative ease of change which macros provide. Certainly as our experience with the Emulator grows, we will find areas for improvement.

The macro approach simplifies at least one such area. The chief alternative to the macro approach, one which is attractive for future use, is an English like language. Unfortunately however, this choice possesses as disadvantages (for the initial system) the negatives of almost every element cited for the macro approach.

Macros defining the particular test (via parameter lists) are assembled into table structures which serve as the data base during test execution and provide system specialization control information for use during test initialization. The macro language is divided into four logical divisions which functionally correspond to those of COBOL. Thus, the Identification Division consists of user specified information for unique identification of a test run or series of runs. The Environment Division defines the virtual terminal population being emulated as well as user specified message volume and scheduling constraints. The Data Division provides the templates necessary for query generation and response analysis. Finally, the Procedure Division defines the information necessary to proceed through a transaction. That is, the decision information necessary to decide "what to do next". For example, what query to send next based upon an analysis of the last response received.

Test initialization takes the linked HCEE programs and tables (on tape in self loading format), performs basic communication startup and housekeeping functions, and dynamically generates certain table entries from the data supplied at test generation time. In addition, the first message to be sent across every active line is generated and scheduled.

During test execution, HCEE generates, transmits, and logs queries and receives, analyzes, error codes, and logs target system responses. Although a limited amount of controlled console operator intervention during test execution is acceptable, the execution phase is capable of running with no outside intervention and with no call on external data storage except for logging purposes.

Following termination of a predetermined test interval supplied by the user, the test result analysis phase is initiated. During this phase the Log Tape is sorted into a variety of sequences (as called for by the user specified reports) and the report generation sub-programs are executed using the log as input, to produce target system performance reports for diagnostic and efficiency measurement purposes.

All four phases of HCEE may be executed contiguously on a load and go basis, or they may be broken into three parts: Generation, Initialization and Execution, and Result Analysis.

Basic facilities

HCEE incorporates the following features:

a. Control
   1. Specification by the user of desired test interval length, and line transaction volume as a time dependent function.
   2. Specification by the user of both the real and virtual (terminal) hardware environment in terms of the number, line distribution, and type of terminals to be emulated.
   3. Specification by the user of context dependent, scheduled time intervals to be applied in initiating queries that represent the continuation of transactions.
   4. Specification by the user of the syntactic and semantic rules which establish the relationship between target system responses received and the next call generated from the same terminal. These rules are supplied as logical table structures which can be probabilistically weighted to ensure the generation of a prescribed transaction mix, and the production of a statistically viable sample of certain expected operator behavior patterns, as (for example) "think time" to absorb received information.
   5. Specification by the user of errors to be introduced into the transmitted data stream either in accord with statistical measures of type and frequency or continual generation of specific errors.

b. Query Generation
   1. Generation of fixed or variable length (format) queries with dynamic selection of key user specified parameter fields.
   2. Dynamic selection of the appropriate query to generate within an interactive context defined by the user in a transaction syntax definition.

c. Response Analysis

   Identification of fixed or variable length target system response types fixed keyword or parameter matching search methods.

d. Error Detection
   1. Dynamic marking during test execution of all response errors. These errors include invalid (unrecognized) target system responses as well as valid hardware and system error conditions, such as line associated errors.
   2. Dynamic marking during test execution of all calls which are generated *after* their scheduled transmission time, distinguishing

a deterioration in HCEE performance from a reduction in Target System efficiency.

e. Reporting (and result analysis)
Post mortem production of a comprehensive set of HCEE test performance analysis reports including such critical measures as response time graphically plotted as a function of transaction volume, polling delay (interval measuring operator request to actual transmission) as a function of volume, HCEE service time by message type, etc.

Emulator program components

HCEE incorporates various fixed and generated components (elements). Each element, along with a brief description of its function, is presented below.

1. Executive—The central element. It is responsible for interrupt management and time slice allocation.
2. Terminal Operations Subsystem—The generative and analytic component of HCEE. It is composed of a fixed element, the Diagrammer; and generated elements, the Query-Response Lists, Vocabulary Lists, and Transition Diagrams. The Diagrammer traverses the Transition Diagrams using the Query-Response Lists in conjunction with the Vocabulary Lists for Query generation and the Query-Response Lists alone for Response analysis. The balance of the paper will concentrate on this subsystem.
3. Transaction Scheduler—The element which provides the time increment Δt (relative to current time t) at which to initiate a transaction for a particular terminal. The decision of what Δt to produce is based on user supplied volume constraints and a random number generator. The Transaction Scheduler is activated by the Communications Subsystem.
4. Communications Subsystem—This element is responsible for transmission and reception of all traffic to and from the system under test. The Communications Subsystem is triggered by the Executive Routine following an interrupt from the multiline controller.
5. Path Selector—The module which has the final responsibility for selecting which query to send. It chooses the query on a user specified percentage basis from a population of queries selected as a result of analysis by the Diagrammer.
6. Console Routine—The element which is responsible for outputting system messages to the console and allowing the console operator to input action requests.
7. Buffer Linker—The element which has the responsibility for linking buffers of a message together and activating the Diagrammer in the case of a fully linked Response, or the Communications Subsystem in the case of a fully linked Query.
8. Message Logging Routine—The element which logs to magnetic tape all transmissions between the system under test and HCEE

Figure 1 illustrates the relationships between the elements of HCEE.

The key

Before continuing with the narrative, it is advantageous to diverge slightly and define some essential terms.

1. Query—Any message generated and transmitted by HCEE regardless of initiating impetus. In other words, even if the message in question is an answer to a request from the target system, it is considered a query.
2. Response—Any message received by HCEE regardless of whether solicited by HCEE or not.
3. Transaction—A sequence of messages (made up of queries and responses) which perform an iden-

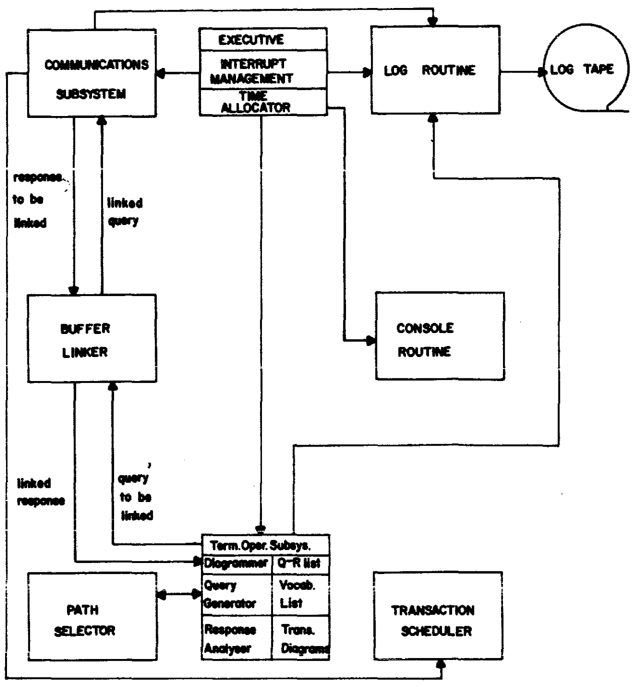


Figure 1—Basic elements of HCEE

tifiable application function. For example, a transaction involving a sale might consist of:

a. How many widgets in stock (query)
b. Ten widgets in stock (response)
c. Sell 10 widgets to ABC Co. (query)
d. Confirm 10 widgets to ABC Co. (response)

4. Transition-State-Diagram—A nodular structure providing finite reference points for states of the system. A completed action on an object may be considered as a state of being of that object. There are indicators and conditions associated with that state and there is an expectation of one or more of a finite number of future operations which might occur on the object as a result of what has already taken place. For example, consider a ball at rest on the ground. It's state of being may be defined as "at rest." This state carries with it information, some of which may be indicators, some descriptors, some measures of expectancy of future events. An example of an indicator may be recognition of the fact that the ball is stationary; a descriptor might be that it has zero kinetic energy; expectance, that it will be thrown up in the air or that it will be thrown against a wall. Any other occurrence is an error or failure of the system involved. Further, consider that at the moment anything happens to the ball, it is started on the way to a new state of being regardless whether what happens is expected or not. If we now define the state of being as a "node," the process of determining which of several possible futures has occurred as execution of an action routine, and completion of that process as arrival at a new state of being (node), then we have explained the concept behind the opening description as a "Nodular Structure." All that remains is to emphasize that all possible relationships are predetermined by the user. Should he make a mistake in defining a relationship, or if he leaves out a relationship, the system will be immediately aware of the mistake. In that sense the process is a heuristic one and allows checkout of the logical dependencies between information.

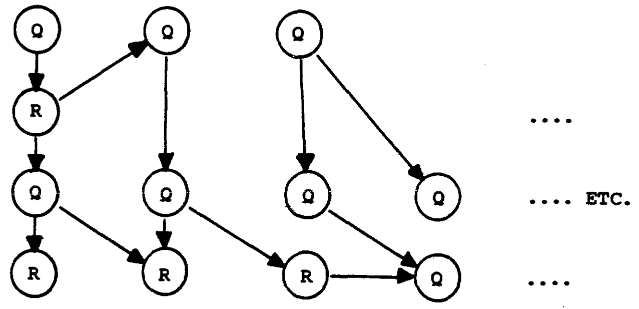
Terminal operations subsystem

The Terminal Operations Subsystem is the heart of HCEE. It contains the prime processing components and is responsible for the generation of queries, checking of responses and maintenance of transaction flow within HCEE. It provides a user with the ability to drive his simulated system over a sustained interval (which may be measured in minutes or hours), without creating an extremely large, highly specific test case library; with surety that he will exercise every path through the target system that he can conceive of and describe, including error paths. Moreover, the user is assured that events within the network will occur in a timely manner, in accord with volume and operator (or system) reaction time constraints defined by him.

The test transaction structure of HCEE is in the form of directed strings of items (Queries and Responses) which permit an exceptionally complex set of relationships to be described. The simplest variation is a one dimensional sequence such as:

$$\begin{tikzcd} Q_1 \arrow[r] & R_1 \arrow[r] & Q_2 \arrow[r] & \cdots & Q_n \end{tikzcd}$$

Complex networks which introduce choice may be constructed similarly:



For response analysis purposes, it is presumed that the paths emanating from a Query Node represent all possible responses to that Query Node (including a default or error state) and conversely that the paths from a Response Node represent all possible queries generatable as a result of the response. The Query/Response mechanism may be further improved by assigning selection probabilities to various queries so that queries are generated roughly with the frequency that they are expected in the actual system. This ability has the added advantage of facilitating experimentation to determine the effect on a system of different message type loading within a specific overall volume.

Test data base

The Test Data Base consists of a collection of transition diagrams (Transaction Logic Network) similar to those just described, which reference a file of user created skeletal message descriptions known as "Query-Response List". The Query entries within the Q-R Lists may in turn reference a set of key word strings or "Query Vocabulary List" which supplies variable word

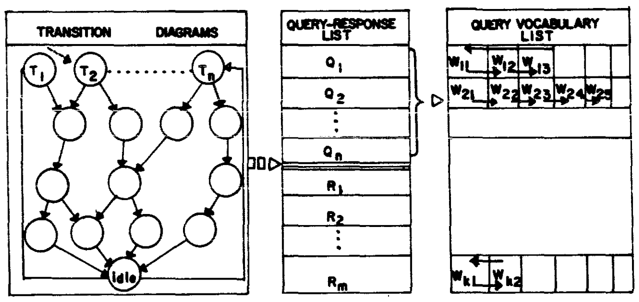


Figure 2—Relationship of transition diagrams to query generation tables

values for inclusion in generated Queries. The relationships are illustrated in Figure 2.

○	: : : node = state defined by the transmission or reception of a specific message type (nodes are either Q or R type where Q = query and R = response).
↓	: : : path = implicit execution of an Action Routine (query generator or response analyzer) necessary to move from one node (state) to the next as directed by the path.
T_n	: : : first node for transaction$_n$
idle	: : : transaction termination node which corresponds to a return of the terminal involved to an idle state.
Q_n	: : : Format Template for query n which is used by the query generator to produce the actual message.
R_m	: : : Identification template for response m which is used by the response analyzer.
$W_1 \ldots W_k$	Variable length word strings (circular linked lists) 1-k which serve as lexical source material for query building, i.e., varying data elements such as part numbers, which are inserted into defined fields in a query template to create a complete query ready for transmission.

The transition diagrams themselves are represented by a set of variable length records, one for each Node in the network. They are created during the test generation process from user supplied information. The Query Node referred to in Figure 3 is the place at which a particular Query is generated and a subsequent response is analyzed. Upon positive identification of the response, control passes to the proper Response Node for preparation of the next Query. Non-identification of a response, indicating either a failure somewhere within the system or a mistake in describing the system by the user, causes control to pass to the error routine associated with the Query Node. HCEE supplies a single common error routine. The user is free to replace it or to add others as he wills. Query path selection using the associated probability measure, and calculation and assignment of the required intermessage interval, are done by the Path Selector subroutine of the Diagrammer.

```
NODE TYPE = QUERY
QUERY-RESPONSE LIST POINTER
NUMBER OF PATHS EMANATING FROM THIS NODE
ERROR ROUTINE

     TARGET NODE (R1)
     TARGET NODE (R2)
          .
          .
          .
     TARGET NODE (Rk)

NODE TYPE = RESPONSE
QUERY-RESPONSE LIST POINTER
NUMBER OF PATHS EMANATING FROM THIS NODE

     TARGET NODE (Q1)
     INTERMESSAGE INTERVAL (Time of delay before
                          transmission of this QUERY)
     FREQUENCY OF USE (A probabilistic weighting factor
                      to assure selection among Queries
                      based on real system expectation)
     TARGET NODE (Q2)
     . . . . . . . . .
     . . . . . . . . .
     . . . . . . . . .
     . . . . . . . . .
     . . . . . . . . .
     TARGET NODE (Qk)
```

Figure 3—Record structure for transmission diagrams

Query-response list

The Query-Response List is acted upon by programs called Action Routines, which are executed by the system while it is traversing the Transition Diagrams. There are two basic types of Action Routines; the Response Analyzer and the Query Generator. The Response Analyzer determines which Response Node to activate. That is, it decides which Response type of a number of possible Responses has actually arrived. The Query Generator concatenates the fields specified in the Query-Response List with values extracted from the Vocabulary List so as to create complete messages from the skeletal formats of the Q-R List.

Query-vocabulary list

Creation of a wide variety of queries from a single Query skeleton is made possible by the Query-Vocabulary Lists. Included in a Query skeleton record are a series of field descriptors which define fields as constant or variable. Variable field identifiers contain pointers to vocabulary list records containing a series of possible values for the particular field. Normally the values will be selected sequentially on a round-robin basis. The technique allows the inclusion of deliberate data errors to check data handling characteristics of the Target System. Figure 4 illustrates a vocabulary list field.

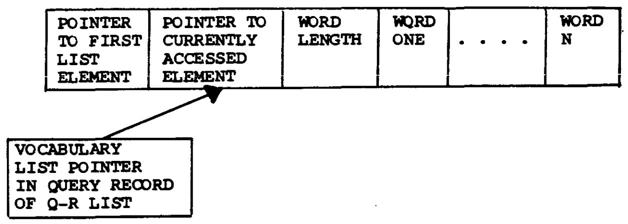

POINTER TO FIRST LIST ELEMENT
POINTER TO CURRENTLY ACCESSED ELEMENT
WORD LENGTH
WORD ONE
. . . .
WORD N
VOCABULARY LIST POINTER IN QUERY RECORD OF Q-R LIST

Figure 4—A vocabulary list entry

Sample network

The diagrams of Figure 5 give a pictorial representation of a very simple one transaction network. In this illustrative example, the first query of the transaction will be "DO YOU HAVE" and then one of a set of items listed in a Vocabulary List. There are three possible responses, "YES," "NO," or "REPEAT MESSAGE". In the case of the first two responses, the transaction is terminated. In the case of "REPEAT MESSAGE", 50 percent of the time a new query will be sent and 50 percent of the time the transaction will be terminated.

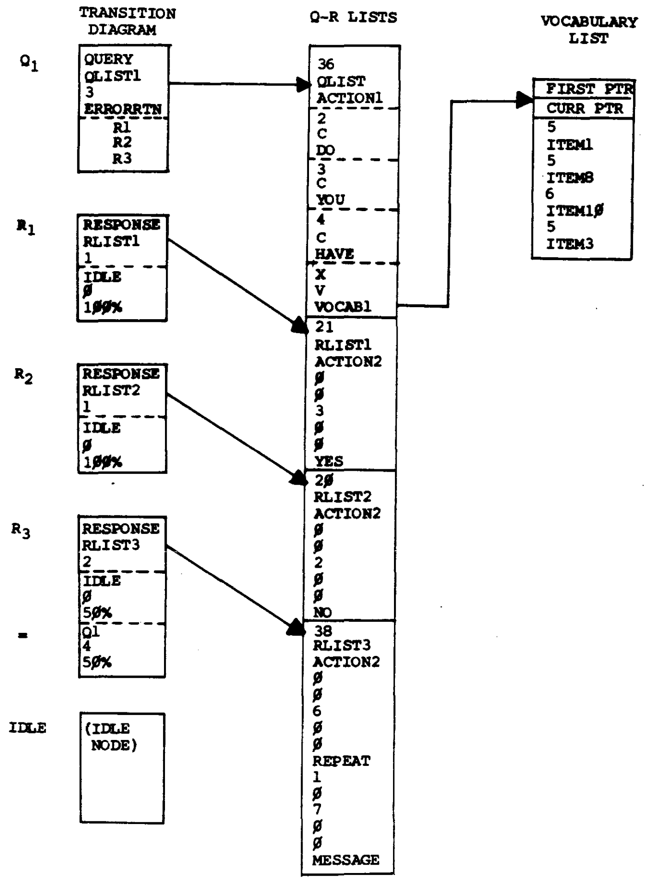


Figure 5—A sample transaction network

Diagrammer

The Diagrammer is the central control routine which coordinates the activity of the Response analyzer and Query Generation Action Routines. At any point in time, for a given terminal, a set of transaction continuation possibilities exist. The set consists of those paths emanating from the current node address of that terminal. Earlier it was stated that recognition of a particular Response led to generation of a specific Query. Actually there may exist a number of valid queries. When selection of a single Query from a population is needed, control passes to the Diagrammer, which retrieves the current node address and in turn activates the Path Selector, which may apply a uniformly distributed random number generator and/or path frequency information present in the node record, to select a single query generation path. Responses are processed and identified by a series of comparisons rather than selections.

Path selector

The prime responsibilities of the Path Selector are selection of the next Query to generate within a given transaction context, and calculation of the time at which the message should be transmitted. We must distin-

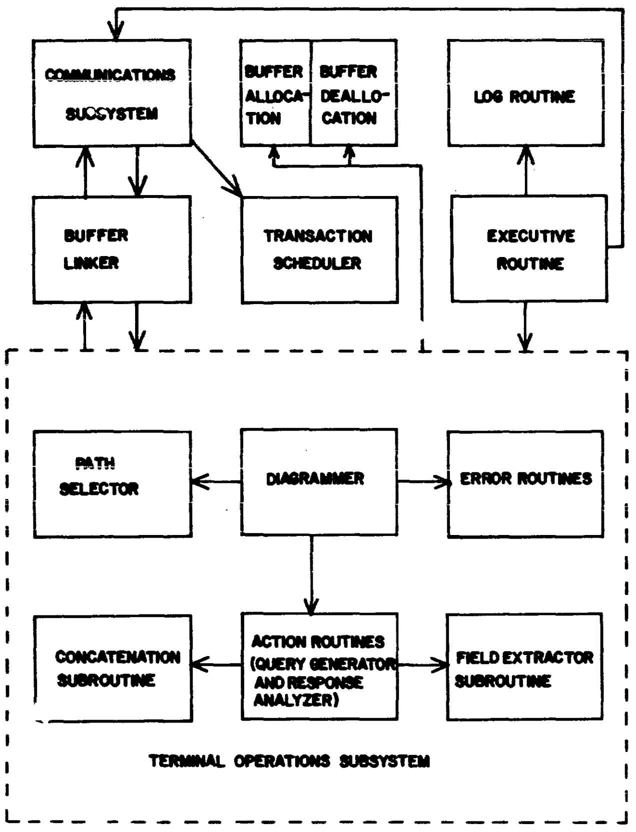


Figure 6—Relationship of terminal operations subsystem to balance of Honeywell communications environment emulator

guish two conditions, each with its own scheduling requirements; initiation of a transaction and generation of a query within an on-going transaction.

Every line or line group (defined as a number of lines with completely homogenous initial transaction requirements) has one or more transaction initiation nodes (Node ∅ for Query initiate transactions—Null Node otherwise). Each possible transaction path for a given line or line group emanates from one such unique node. Each potential transaction carries a usage frequency (provided by the user) which the path selector uses as a selection constraint. Scheduling for Transaction Start activity is not done by the Path Scheduler, but rather by the Transaction Scheduler. This latter routine selects inter-transaction intervals based upon transaction volume constraints imposed by the user.

Query selection for on-going transactions has already been described. Scheduling for such queries is done by the Path Selector. This is accomplished by adding a user defined inter-message interval to the arrival time (from system time zero) of the activating response, and posting the result as the scheduled transmission time.

CONCLUSION

We have just described a technique for providing a real time checkout facility to a programmer or software designer writing a communication based system. The design is flexible, modular and expansible. The program is expected to be ready by the fourth quarter of 1969.

Development of New York City's geographic data network

by ROBERT AMSTERDAM

Office of Administration, Office of the Mayor
New York City, New York

INTRODUCTION

New York City has begun the coordinated development of a network of information systems which will assist all agencies of the City government in exchanging information needed for routine operations, planning and analysis. The approach used here places operational requirements in the paramount position. That is, the methods by which agencies can receive, use and transmit data vary widely and depend largely on the services each agency is required to provide. Thus, the concept frequently proposed, of a massive, centralized urban data bank was found to be inadequate. Instead, New York is developing a series of systems using various methods for data handling as determined by the requirements on each system. The heart of this network is a system which stores and transmits current information on the data elements which are used by agencies throughout the City government. These are elements relating to the basic geographic environment of the city; the land, streets, buildings and location of public facilities.

This paper discusses the factors which determined the overall design, the principal components of the network and the areas in which current work is progressing.

Considerations in designing an urban information system

Rationale

The reasons for developing a coordinated urban information structure are well documented.[1] To review them briefly:

1. Many agencies of local government need the same information but their resources for gathering and maintaining these data vary widely. For example, both assessors and redevelopment planners must evaluate the condition of existing structures in relation to the neighborhood in which the building is located. The assessor's department, because it has a continuing responsibility to know the use of every piece of taxable land in the city, has developed detailed records to maintain data on every parcel and systematic procedures to update these records. The development planner is concerned with only a small portion of the city, his concern is immediate and he generally does not have the time or facilities to develop the information base he needs.
2. Each agency generally needs its data organized in a different fashion. Again, using the example of the assessor and the redeveloper, the former generally must handle his files sequentially to be sure that no parcel has been overlooked. The developer needs to examine parcels in a block or neighborhood orientation in order to determine what portion of the community is best suited for redevelopment.
3. Many agencies in local government organize their data according to district lines determined by the particular service they are providing. For example, school districts are drawn to meet the distribution of school-age population and the locations and capacities of schools. Sanitation districts are drawn with consideration for the location of incinerators and the garaging of collection trucks. It should be noted that in many localities the different services are performed by administrative units with overlapping jurisdictions. Thus a suburbanite frequently finds his water is provided by one administrative unit, schools are administered by a different unit, police by a third unit and fire protection

by a fourth. For this reason, data which is collected and summarized by one agency is, in aggregate form, useless to another agency.
4. Agencies frequently collect and store duplicate information. Because of the reasons cited above, or because one agency is not aware of the data which another one possesses, there is much duplication of data gathering. Very often this involves extra burdens on the public. For example, in New York City, if an individual acquires a piece of property, he may be required to register this fact in up to four different offices.
5. In many cases one agency can easily obtain information which several other agencies need but have great difficulty in acquiring. For example, many agencies in New York need information on structural details of buildings which can be obtained most easily from the architect when he files his plans.
6. Files which have significant relationships in common frequently cannot be compared because of differences in the schedules of updating or the method of data acquisition. In many cases an agency must be cautious in combining data from two files because they cannot determine whether the same standards were used for both.
7. There are many requirements to analyze or summarize data contained in files which are maintained manually. The size of the files makes it a practical impossibility to summarize their contents.

Obstacles to automation

In the past there have been a number of obstacles which have made it unfeasible to consider automation of many local government files. Large random-access storage devices, reactive terminals and graphic display equipment have started to change this balance but for most cities these obstacles are still appreciable:

1. City records have a long life of usefulness. A distinctive characteristic of most local government records is that the data is of interest for many years after it is acquired. In industry, by comparison, if a record is more than a year old it would probably be of value to no one. City records of building plans, title transfers and birth certificates, as examples, are maintained permanently. Any plan for record automation must adequately justify the cost of permanently storing records which may only be searched sporadically if at all.
2. City files are large and only grow larger.
3. The records tend to be long and complicated. They will generally contain many types of data, frequently in a complex pattern that is difficult to systematize. When automation is considered, decisions are generally made to leave out some data elements that are difficult to classify or not immediately useful. In many cases the elements omitted are later found to be necessary and are acquired for additional cost that could have been avoided.
4. Some data must be protected by the agency acquiring it. Government agencies collect much information which is generally agreed to require confidential handling. As one example, the number of employees in a firm could be valuable information to a competitor. More familiar is the information on individuals collected by the U. S. Census Bureau, departments of social services and police. Where sensitive information is involved, the originating department will always aggregate the data to the block or district level before releasing it to other agencies.
5. City files are hard to match because of problems of identification. In matching addresses, for example, a structure may be known by more than one house number or the street name may be commonly spelled in a number of different manners. In addition some agencies do not identify a property by house number. The assessor, for example prefers to use lot numbers. The developer, who may be acquiring or subdividing lots, may use a series of parcel numbers.

Conclusions from earlier work

It is important for anyone who works in this field to keep abreast of other efforts. Since much of the activity in developing urban data bases has been federally funded, hard-learned lessons have been very well documented.[2] The following advice is particularly worth reiterating.

1. A grid system (x, y coordinates) is essential in any urban information system in order to provide a fixed point of reference. It is also useful in calculating distance, direction, area and proximity.
2. Data should be collected at the parcel or building level wherever feasible and aggregated when necessary to meet specific uses.
3. Data should be captured at its source and

transmitted to a data processing center by the source department.

4. Data files should be maintained by the source department, if possible as a by-product of their routine operations.
5. No organization can be expected to contribute to an activity on a continuing basis unless the organization receives some benefit in return.
6. "Systems for urban information handling should not be designed to serve planning alone, but should be able to provide data to other agencies of local government."[3]

Data processing environment in New York City

EDP operations of the City of New York presently involve eighty-nine computer installations including those in the courts, schools and colleges. Twenty-four computers are used by agencies directly responsible to the Mayor. These are used primarily for batch sequential processing of tape files running to 21 reels. The computers are located at sites throughout the City Hall area. Many agencies with large data processing requirements have multiple installations.

As might be surmised, the volume of data processed by the City is vast and any attempt to centralize the data of general interest in one place would present problems of administration and coordination beyond those already discussed.

There are two factors in New York's administrative situation which aid the objective we are concerned with. First is the fact that all local agencies gathering geographic data on New York City are responsible to the Mayor. In many other communities there is a division between city and county functions which makes coordination of files an impossible consideration. Very frequently property assessment is a county function as in Cook County, Illinois, where Chicago is located. The second factor is that Mayor John Lindsay has re-organized many departments in new superagencies. This has started processes of centralization, cooperation and re-examination of existing data resources and needs. These processes have greatly aided the coordination of data sources on a city-wide basis.

Scope of New York City's geographic information system

General

New York City consists of five counties containing approximately 48,000 blocks separated by 9,000 different streets. The blocks are divided into 830,000 lots which hold over 900,000 buildings. City government responsibilities covering the streets, the shoreline, the land and the buildings is spread through more than twenty separate agencies. Figure 1 presents New York's geographic data network as it might look at the administrative level when the network is fully developed. All agencies shown will be routinely exchanging data with a central Geographic Information System (GIST). In addition, many of the agencies will be exchanging data with each other. The lines shown between agencies indicate the most probable links that will be required. Most of these links will probably consist of manually transported magnetic tapes. In some cases batch teleprocessing and on-line communication may be required

Four components of this network will be described in this discussion: the GIST system and the three agencies whose data-gathering activities will furnish the principal support to the network.

GIST is intended to serve as the fulcrum or central switchboard for the network. It will facilitate coordination and exchange of data among agencies through various types of standardization and correlation. For example, it will be able to identify the building corresponding to a known tax lot number; it will recognize when two addresses on different streets identify the same building. These are typical of the problems which restrict coordination and use of many records at present.

GIST will maintain files on data items of wide general interest. Its files will be maintained and updated through the activities of operating agencies as a by-product of their own data processing. The GIST System

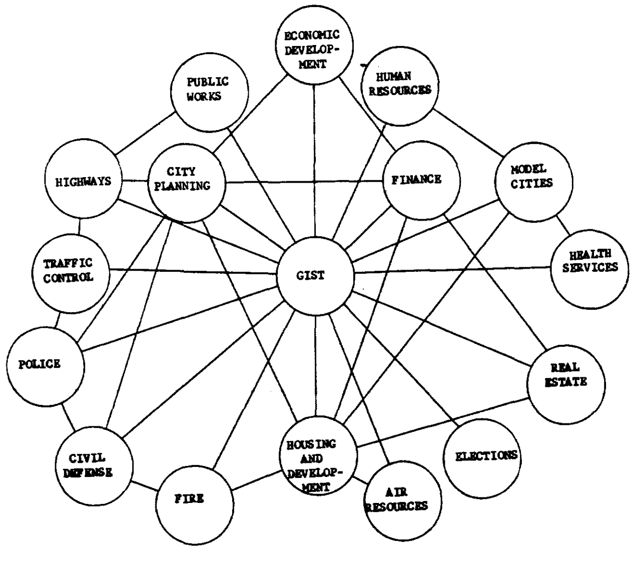


Figure 1—Conceptual view of geographic data network

will function as part of a municipal EDP Service Center, with capabilities for processing geographic data to satisfy various requests: simple queries, complex analyses, matching and checking of addresses, automatic generation of maps and overlays. It may, at a later stage of development, serve the general public.

The GIST files will be restricted to current-status data. The files of the operating agencies will contain the supporting historical records and detailed information which generally are used only by the agency which collects them.

Near the center of the network are three agencies which generate more than 90 percent of the data which other agencies are interested in obtaining: City Planning, Finance Administration and Housing and Development Administration.

The Department of City Planning, which is the arbiter of zoning, determines the pattern of land use in the City, i.e., what kind of buildings may be erected, their size, their use. It also establishes where public improvements will be located, such as schools, parks and highways. To support their decisions the Department conducts research on economic activities, residential trends and population movements. They assist the Bureau of the Census in planning the census-taking in the City. Most of their research results are available for general circulation, but partly because of the effort of correlation needed to utilize results of different studies, it is generally preferable at present to focus on particularly critical areas of the City at any one time.

One significant file which City Planning maintains is the Address Coding Guide. Similar files are maintained by many localities to meet the requirements of the Bureau of the Census. The file relates house addresses to census blocks, tax assessors' block numbers, health areas, police precincts, planning districts and other zones of significance. The file is used for locating buildings and coding addresses so that data can be summarized geographically for statistical purposes. Use of the file as an index is restricted by the fact that it is maintained on tape (five reels) contains approximately 54 million characters. Almost all of this file will be duplicated in the central GIST system.

The Finance Administration combines three departments: Registrar, which maintains the records of property ownership, Property Assessment, which determines the fair values of land and buildings, and the Finance Department, which collects taxes on property, water and sewer use, incomes and other sources. These departments have complete and systematically maintained records covering all buildings in the City. It should be noted, however, that there are some significant weak points in their files. For example, when properties are taken by the City, either through condemnation or forfeiture, or when properties are transferred by will or by court action, the title is frequently not recorded in the Registrar's Office. Also, since many property owners pay their taxes through banks or mortgage companies, their identity cannot be precisely determined. Finally, since the assessors are not concerned with tax-exempt properties, the records on these buildings, which include city-owned buildings, are not always complete.

Most of the assessors' more important data are already machine-readable. These include the lot dimensions, building size, building use, assessed value and current owner or mortgage holder. However, property assessment is becoming more sophisticated, making more systematic use of data on convenience to public services, comparisons with similar buildings and character of neighborhood. All of these items are of wide interest and when captured in machine-readable form, would be maintained in the central system as well as in the Property Assessment System. Similarly, the Registrar's records on ownership and mortgages are being considered for computer conversion and these data are also of importance to many agencies.

The Housing and Development Administration consists of five agencies concerned with the various aspects of new construction, relocation and maintenance of the housing stock. Three departments are significant here. First, the Department of Buildings which reviews all new building plans, inspects elevators, boilers and on-site construction and issues permits for demolition, construction and occupancy. Next, the Department of Development which plans and develops publicly-financed housing. Finally, the Department of Rent and Housing Maintenance which supervises rent-controlled apartments and, through various programs, seeks to assure that all housing is adequately maintained.

A major effort is now in progress to develop a Housing Data Bank which will enable the many operations within HDA to draw on each other's resources. This is being developed in conjunction with several major applications which should greatly improve the effectiveness of housing maintenance and inspection programs. The Housing Data Bank, as presently planned, would reside in a combination of fast and slow random-access storage. It would contain approximately 3.1 billion characters. Only a small portion of these items are of general interest outside HDA.

It may be noted from the above that the different agencies are, for some programs, interested in all parts of the city while for other activities their interest is centered on specific areas or particular types of buildings. There is no reason therefore, to expect that the City's automated files will be equally complete for all data being collected nor do they need to be.

Current activities

Several components in the data network are presently in development. The Police Department's SPRINT System, using on-line terminals to locate requests for help and dispatch police cars and ambulances, will be in operation by the end of 1969. The Model Cities Committee, which is developing detailed data bases for the Model Cities areas has been using complex matrices to determine the optimum mix of renewal projects for a neighborhood. The Housing and Development Administration, as mentioned earlier, is designing several major applications around a large Housing Data Bank.

The remainder of this discussion will be on the GIST System, its overall design and current development. GIST is being developed by the Office of the Deputy Mayor-City Administrator, Dr. Timothy W. Costello under the supervision of Dr. E. S. Savas, Deputy City Administrator.

Overall design of GIST system

1. Files
   GIST will be supported by three principal files. The items planned for inclusion in each are shown in Tables I through III. The source shown next to each item identifies the agency which will provide and maintain that item. The method and schedule of maintaining each item in the GIST files will be established for the convenience of the source agency. Briefly, the files are as follows:

   a. *The Street-Name File* will contain the legal name and commonly used names for every street. In addition, street code designations in wide use, such as Police and Buildings Departments, will be included. The GIST system will generate a five-letter code for each street for internal identification and sorting.
   b. *The Block-Face File* will contain one record for each block side. Thus, for a normal rectangular block, four records would be maintained. Each record will contain locational data relevant to the block or block face. This will include tax block number, census block number, range of house numbers, direction of traffic and geographic coordinates at each corner of the block. These coordinates have been determined through work by the Tri-State Transportation Commission using aerial photos. Accuracy is to the thousandth of a mile — that is, to within five feet. Other data included will be block adjacencies. These describe the spatial relation of the block face to its neighbors, e.g., around the corner, across the street, directly opposite, etc. These data will simplify many types of districting and routing applications where it is necessary to know the orientation of a block side.
   c. *The Building/Lot File* will contain one record for each lot, major structure and owner. If a structure occupies two or more lots, there will be added records for the extra lots. If there are two or more major structures on one lot, there will be an added record for each additional structure. If distinct portions of a lot are held by different owners, particularly if part of the lot is publicly owned, there will be a separate record for each owner.

Each file will be related to the others, that is, Table I will serve as a directory to Table II, Table II as a directory to Table III.

Table I Street-name file

There will be one record in the Street Name File for each street name in common use.

Item	*Source*
Borough	City Planning Department
Common Name	Being developed
Official Name	City Planning Department
Building Department Street Code	Building Department
Police Department Street Code	Police Department Sprint System
GIST Code	Being developed

Table II Block-face file

IDENTIFICATION DATA

Item	*Source*
Record Identification	Being developed
Borough	City Planning Department
Tax Block No.	City Planning Department
Tax Block Suffix	City Planning Department
S - Street Name	City Planning Department
t	
r - Street Suffix	City Planning Department
e	
e - Street Ending	City Planning Department
t	
City Sectional Map No.	City Planning Department
Census Tract	City Planning Department
Census Block	City Planning Department
Census Block Suffix	City Planning Department
Type I Adjacent Block sides	City Planning Department
Type II Adjacent Block sides	City Planning Department
Type III Adjacent Block sides	City Planning Department
DISTRICTS	
Health Area	Dept. of Health
Health District	Dept. of Health
Police Precinct	Police/Sprint
Fire Company	Fire Department
Sanitation District	Sanitation Department
Hospital District	Police/Sprint
LOCATION	
Street Segment Identification	Being developed
Number of block sides on block	City Planning Department
Block Side No.	City Planning Department
High House No.	City Planning Department
Low House No.	City Planning Department
High Intersecting Street	City Planning Department
Low Intersecting Street	
High block side No.	City Planning Department
Low block side No.	City Planning Department
Geographic coordinates:	City Planning Department
High Corner: XH	City Planning Department
YH	City Planning Department
Low Corner: X_L	City Planning Department
Y_L	City Planning Department

Each record will be as complete as possible for information concerning building description, land description, ownership and current permits and violations. It will be the user's responsibility to determine how accurate, complete and current the file is for the items he plans to use.

2. Operation
The capabilities, functions, and plan of operation for GIST are shown in Figure 2.

The user, representing any City agency, will enter his data into processing together with instructions which define the format of

Table III Building/lot file

There will be one record in this file for each building, lot and owner.

Item	*Source*
OWNERSHIP DATA	
Owner/Manager's Name	Finance Administration and others
Owner/Manager's Address	Finance Administration and others
Mortgage Holder's Name	Finance Administration and others
Mortgage Holder's Address	Finance Administration and others
IDENTIFICATION	
Principal House No.	Finance Administration
Range of Numbers for House	To be developed
Tax Lot No.	Finance Administration
LAND DESCRIPTION	
Frontage	Finance Administration
Depth	Finance Administration
Irregular Plot Code	Finance Administration
Total Area	To be determined
Present Land Use Code	City Planning Dept.
Special Land Features	To be developed
Zoning Classification	City Planning Dept.
BUILDING DESCRIPTION	
Year constructed	Finance Administration
Type of construction	Buildings Department
Frontage	Buildings Department
Depth	Buildings Department
Total area	Buildings Department
Floor space (sq. ft.)	Buildings Department
No. of stories	Buildings Department
Latest alteration (date)	Buildings Department
No. of buildings	Buildings Department
Private garage (res.)	Buildings Department
No. of establishments	Buildings Department
No. of dwelling units	Buildings Department
PERMITS AND VIOLATIONS	
Demolition permit (date)	Buildings Department
Construction permit (date)	Buildings Department
Certificate of Occupancy (date)	Buildings Department
Crane permit (date)	Highways Department
Violations (no.) :	
Fire Department	Fire Department
Buildings Code	Housing and Development Administration
Health	Health Service Administration
Sanitation	Environmental Protection Administration
Air Pollution	Environmental Protection Administration

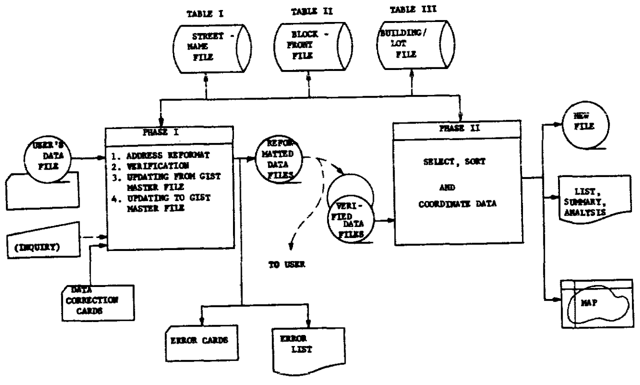


Figure 2—GIST—Geographic information system

his data fields and indicate what processing is to be done. For regular users, standard instruction routines will be provided.

If the input records contain house addresses, the input will first be processed in Phase I. The street names will be verified and, if necessary, reformatted and corrected. A street code will be added to facilitate internal identification and record sorting. A master file such as that in Table I will be required with a record for every street name in common use.

If a street name is not recognized, an error message will be produced by a method which facilitates correction. Subsequent tasks will be determined by the user's instructions. In the simplest case, he may want a new item, such as health area or assessor's lot number, added in a designated field on each record. These items would be obtained from Tables II or III. The end product would then be a tape with the same number of records and in the same format as the original but with the requested data added.

The user's initial input could be identified by means other than house address. For example, data could be identified by tax block, by street, by health area or by geographic coordinates. It will be possible to request data concerning buildings within a given set of geographic boundaries or within a circle of given radius from a focal point.

More complex tasks will be supported as they become feasible: records may be combined or split on the basis of master file data, records may be selectively updated, etc. The variety of tasks can be gauged from the items included in the GIST tables.

Another series of tasks will be required so that input data can be used to update the master file. This will involve splitting and consolidating existing records. Initially, the files will be stored on tape. When the system is fully implemented, the files will be located on random-access devices. It is expected that eventually the system will be modified to accept data and queries entered from on-line consoles.

The output of Phase I processing will be in the form of a tape. Dependent on the user's instructions, this will be returned to the user for further processing at his own installation or it will be entered in GIST Phase II.

Phase II will select and sort data from designated input files which have been through Phase I processing. These can be one or more files which the user has supplied, either from his own agency or from other cooperating agencies or from the GIST files. Various manipulations and data transformations could be performed. Output would be in the format specified by the user. Tape, cards, printed output, or maps would be produced, at the option of the user.

Each task described will probably require a number of programs to accomplish, and a set of user instructions will be required to specify the various tasks. RPG and natural language are among the instruction vocabularies being considered.

Current development of GIST

Full implementation of the GIST system is expected to take five years. However, two capabilities planned for GIST will be available by the end of 1969. These should assist the solution of specific information handling problems which are present in many City agencies.

1. *Standard Address Generator*
   The first is a set of programs which will accept any file of machine-readable records that contain street addresses and reformat and standardize those addresses for computer handling. If the house-number field overlaps the street-name field, the two fields will be separated and reformatted for subsequent processing. The street name will be verified and minor spelling errors corrected. If the street name cannot be verified, an error message will be produced. In the final program output, for each record successfully

processed, a field will be added to the start of the record containing the reformatted identification data. Included in this data will be a five-letter street name contraction.

This output tape will be available for sorting, matching or other processing that the user chooses. These programs include the functions shown as steps 1 and 2 in Phase I processing in Figure 2. This application will facilitate the analysis of data in street address files and permit matching of data contained in address files of different departments which are in incompatible formats.

The programs will operate on IBM 360/30 or larger machines utilizing 16K DOS/TOS operating system with core storage of at least 32 K bytes. It will be a modification of a set of programs produced for the U. S. Bureau of the Census.

2. *Geographic Conversion and Mapping*

The second capability is a set of programs and files which can operate on standardized addresses (such as those obtained through the Standard Address Generator) to provide certain kinds of locational information. It will, for example, provide the health area, tax block or census block number for a given address. It will provide the approximate geographic coordinates for an address, and it will have the capability of drawing maps.

Thus, for example, it will be possible to generate a map of registered voters showing the number of registered voters in each building. It would prepare maps showing the distribution of ambulance calls in a borough. It would also be possible to generate a map showing the addresses and locations where new construction permits or demolition permits had been issued. It could select and map any information which could be identified by street address or located to a geographic area.

The output could be a tape with the desired locational data added to each record. It would be possible to use this tape selectively for plotting and mapping. Figure 3 shows a map section which has been drawn by a computer as a demonstration of the mapping capability. The map is of a section of upper Manhattan. The number, obtained from the Board of Elections list of registered voters, shows only

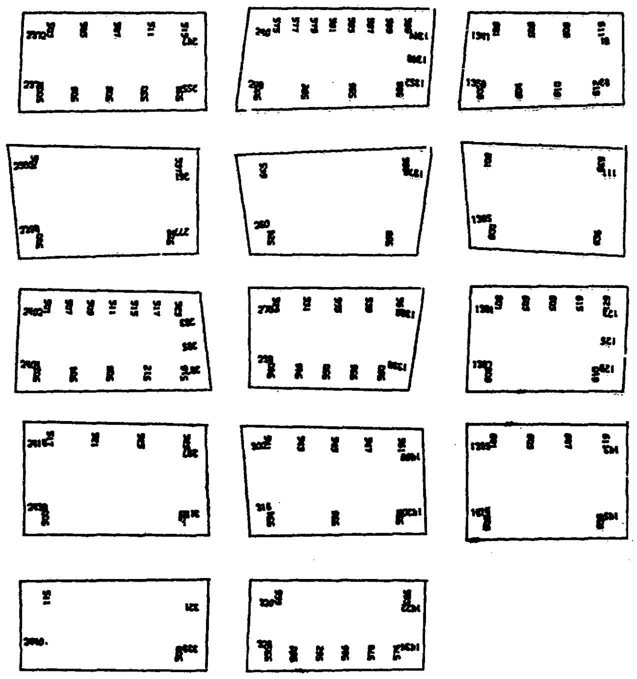


Figure 3—Computer-generated map showing addresses of registered voters

the houses where registered voters live. This capability should have use in a variety of planning and resource allocation activities.

REFERENCES

1 SYSTEM DEVELOPMENT CORPORATION
Report of the APOF conceptualization study
Santa Monica California March 31 1966

2 CITY ENGINEERING DEPARTMENT
An information retrieval system for urban areas
Vancouver British Columbia May 1967
EAST-WEST GATEWAY COORDINATING COUNCIL
The role of locational control in an information system
St. Louis County Missouri September 1967
S ARMS
Map/model system—technical concepts and program descriptions
Portland Oregon February 1968
COUNTY EXECUTIVE
Management information system
Nassau County New York March 1968
METROPOLITAN PLANNING COMMISSION—KANSAS CITY REGION
Integrated information system for urban planning
Kansas City Missouri August 1968

3 METROPOLITAN DATA CENTER
Metropolitan data center project
Tulsa Oklahoma 1966

Requirements for the development of computer-based urban information systems

by STEVEN B. LIPNER

The Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

Since early in this decade urban planners and systems analysts have advocated the development of computer-based urban information systems. Such systems would store detailed data about the environment in which planning agencies and governments operate. They would be organized to lend integration to data from diverse sources, to provide quick preparation of reports and to simplify and automate numerous clerical functions. Many attempts have been made to develop urban information systems with the characteristics mentioned above. Most have been unsuccessful[1] for a combination of technical and organizational reasons. This paper considers some technical requirements for planning information systems which deal with data associated with urban locations. The requirements are developed on the basis of experience in providing a prototype urban information system to the Boston Model Cities program. The next section describes briefly the experience of providing an information system to the Boston Model Cities program. Succeeding sections draw on this experience to develop general technical requirements for urban information systems. A technique for aggregating data by geographic area is presented and its implications for system file structure and utilization are explored.

Information system for the Boston Model Cities Administration

During the spring of 1968, M.I.T. staff members held a number of meetings with members of the staff of the Boston Model Cities Administration to determine how M.I.T. might assist Boston's Model Cities program. One of the major desires of the Model Cities staff members was to see if an urban information system could be used to aid their planning and program evaluation activities. The Model Cities Administration was undertaking a survey which would determine the land use, building condition, and building size associated with each parcel in the Model Neighborhood Area. It was agreed that this data would make an acceptable basis for a prototype urban information system. Model Neighborhood residents employed by the Model Cities Administration were trained in key-punching and prepared approximately 8000 cards, one for each parcel in the area. (For comparison, the city of Boston contains about 100,000 parcels.)

The survey data was input to the ADMINS[2,3] system operating on the time-shared 7094[4] at the MIT Computation Center. ADMINS is an interactive program capable of performing data selection and cross-tabulation. It was designed for use in the analysis of social science surveys, and is best suited to operating on small files of coded or integer-valued data items. It is weakest in the areas of data modification, large file handling, and real or alphanumeric data manipulation.

Initial preparation of the data for ADMINS analysis was judged too complicated and machine-oriented a task to be performed by persons with little computer training. Accordingly the data was prepared for analysis by MIT personnel experienced in programming and in the use of ADMINS. The data preparation was simplified by the ability of ADMINS to accept data in arbitrary codes and formats and by the interactive mode in which it is used. Errors in the data were reported by ADMINS programs and corrected by using the time-sharing system's general purpose editing capabilities to modify the input files.

The analysis of the Model Cities survey data was performed by three groups of people: MIT staff members with substantial computer experience, professional urban planners with little or no prior computer experience, and Model Neighborhood residents with neither computer experience nor extensive formal education. All three groups easily mastered the mechanics of producing desired cross-tabulations, although a natural "fear" of the computer terminal had to be overcome by those new to it.

The response of the planners to the prototype urban information system was both interesting and significant. Although they had been instructed in the use of ADMINS at the terminal, and given freedom to produce reports as needed, the planners preferred to contact MIT or Model Neighborhood personnel, describe verbally the required tables, and have the resulting hard copy delivered to them. Whether this phenomenon was caused by the lack of proximity of the planners to the terminal, by the relatively tedious ADMINS language, or by a basic reluctance of planners to use the computer directly remains undetermined. (Placement of a terminal at the Model Cities office has been planned for some months but has been delayed by various administrative and operational problems.) When the planners have more direct access to a terminal and are provided with a system which, unlike ADMINS, is designed to serve as a true urban information system, it should be possible to determine if experienced planners without computer experience can successfully be trained and encouraged to use a computer as a planning tool. The implications of such a determination are discussed in the next section.

The analytic results produced for the planners using ADMINS were useful, and all agreed that they were pleased with the results of the analysis. The limited computer experience, however, whetted the planners' appetites for more diverse capabilities. These capabilities included:

1. The ability to aggregate data by arbitrary geographic areas such as school districts, without being required to list explicitly every block contained in each area.
2. The ability to produce maps and graphs as well as tables.
3. The ability to merge data gathered by operating agencies and survey research organizations with stored data.
4. More general capabilities for numeric and alphabetic data processing than those provided by ADMINS.

The experiment in computer-aided Model Cities planning has been successful in two senses. First, it provided valuable insights into the capabilities required of an urban planning information system. Second, it introduced a group of planners to computer-aided analysis. In the future these planners should provide valuable data on the mode of man-machine communication appropriate for an urban planning information system.

Requirements for urban information systems

The experimental provision of computer support to planners described in the previous section provided several insights into the capabilities required of an urban information system and the specific features required to implement them. Perhaps the most important capability indicated is that of combining and using in a single information system data from a variety of sources. Special surveys are an expensive and short-lived source of planning data when compared with operational data which must be maintained, often in machine-readable form, by agencies other than the planning department. Operational data from a given agency, in order to be useful to the planner, must be combined with planning survey data and often with data from other public or private operational agencies. Since different agencies often use different identifiers for each parcel, and since the street address is the only common and (presumably) unique parcel identifier, the conclusion is reached that a useful planning information system must deal with parcels identified by street address. Address matching programs[5] have been developed which standardize the formats of street addresses keypunched in free format. They must be included in an urban information system, along with file structures appropriate for the identification of parcels by street address. The need to merge data from differing sources implies the possibility of varying amounts of data describing a single parcel. Such possibilities must be handled by a flexible but efficient data file structure.

A second major requirement of an urban information system is the ability to aggregate parcel data by arbitrary geographic area. This ability is especially important in view of the numerous overlapping administrative and planning districts into which urban areas are divided. Programs have been developed[6,7] which aggregate data into districts by first assigning coordinates to each parcel, and then testing each parcel to see if its coordinates lie within a district. Such programs work but seem suited mainly to sequential storage systems using fast computers. The reasons for this observation and an alternate technique based on street addresses will be presented in the next section.

The importance of graphical display of data to planners was emphasized during the initial work with model cities planners. Any really useful urban information system must produce graphical as well as tabular output, preferably with minimal user description of coordinates, scales, etc. Existing programs and systems[8] are capable of producing a wide variety of graphic outputs. The major problems in applying these to urban information systems are, first, assuring that the outputs they produce are those required by planners and second, integrating the graphic components with data management components to minimize the complexity and cost of producing the outputs.

The area of man-machine communication is one which may be critical to the success of urban planning information system design. The experiment described above produced results which can only be described as inconclusive. However experience in the use of computers by engineers[9] would seem to indicate that the use of computers by persons who are not computer-oriented is greatly aided by the availability of interactive problem-oriented languages. In order to produce definitive results in the area of communication between computer and planner it will be necessary to provide both better terminal access and a problem-oriented language superior in both power and usability to that of ADMINS. The growing presence of planners who have had computer training should provide further assistance in improving man-machine communications.

In re-examining the requirements developed in this section, we find that all except those of geographic aggregation of data, address matching and graphical output would be common to any powerful information system: file structures which allow items to be described by varying numbers of attributes, file structures for rapid data retrieval, and powerful problem-oriented retrieval languages are all provided by many modern information systems.[10,11] Of the required features which appear unique to urban information systems the most significant seems to be that of geographic aggregation of data. Address matching is essentially a preprocessor function and graphic output an important output processor, while the geographic aggregation method will have a significant effect on the cost of many retrieval requests and some influence on internal file organization. For this reason, the next section is devoted to a brief description of an alternative to existing schemes for geographic aggregation of data.

A technique for geographic aggregation of parcel data

The problem of geographic aggregation of parcel data in urban information systems has typically been handled by "point-in-polygon" programs.[6,7] Such programs require that each parcel which is included in the information system be identified by its x-y coordinates. An area for which data is to be aggregated is described as a polygon by specifying the coordinates of its vertices. Each stored parcel is tested by counting the intersections of a ray of arbitrary direction originating at its identifying point with the sides of the polygon. If the count is even, the point (and hence the parcel) is outside the polygon. If the count is odd, the point is inside (Figure 1).

Although the point-in-polygon test is a workable technique for geographic aggregation of data, it poses two problems. First, and less significant is the problem af assigning coordinates to every parcel. This problem is easily solved by representing every street as a sequence of line segments and using the numerical value of each parcel's address first to select the segment containing the parcel and then to define the parcel's coordinates by interpolation between the segment's end points. The second and more serious problem presented by the point-in-polygon technique involves processing time. Since the point-in-polygon technique is a test on one parcel, every parcel recorded by a system must be tested to determine which parcels should be aggregated into a given area. Thus, the technique is ill-suited to systems employing direct-access storage devices which could allow selective access to desired parcel data. Furthermore, the calculations required to determine whether or not each parcel lies in a given area involve one line intersection for each side of the area. On some small computers this calculation may be relatively time-consuming. Thus even if the parcel data base were recorded on tape, the time required to select those parcels in an area could be governed by processing time rather than by the time required to move and read the tape.

Techniques have been suggested[12,13] which, by dividing an urban area into subareas, would reduce the sequential file searching required by the point-in-polygon algorithm. These techniques would require checking of the retrieval area for overlap with pre-established subareas before individual parcels in the

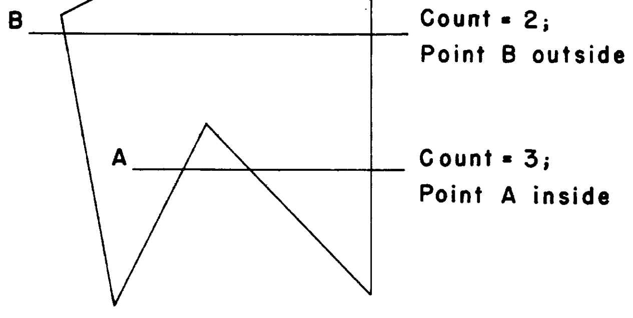


Figure 1—Point-in-polygon test

subareas were examined. If the check showed no overlap, no further examination of the subarea would be required. Otherwise every parcel in the subarea would be checked. The disadvantages of this method are principally associated with the size of subareas. A large number of small subareas requires a large number of overlap tests, while if a small number of larger subareas are used, there will be a large number of parcels requiring point-in-polygon testing included in each selected subarea.

An alternative to the point-in-polygon technique for the geographic aggregation of parcel data was suggested first by Farnsworth[14] and later proposed independently and in more detail by Parsons.[15] The algorithm involves using a map of the street network of the urban area within which new geographic areas are defined. Given a list of the names of the streets surrounding the area of interest, the algorithm produces a list of those parcels within the area. The paragraphs below present an illustration of the algorithm, followed by comments on the map file structure required to implement it.

In considering the map of Figure 2, let us assume we wish to isolate the area bounded by streets A, H, D and E. We first scan the street A until we locate the set of street segments (portions of a street between two intersections) on it between E and H. We then scan street H, marking the segments between A and D, street D for the segments between H and E, and street E for the segments between D and A. Since the list of bounding streets was given in a clockwise direction, we know that blocks inside of the desired area are to its right. If we have recorded the numbers of the blocks to the right and left of each segment, seen facing in the direction of increasing addresses, we may now isolate those blocks inside the bounding streets. To do this we record blocks to the right of segments whose increasing address direction coincides with the direction of the area boundary (street A and E) and blocks to the left of segments whose addresses run opposite to the boundary (streets D and H). Applying this procedure we obtain the list of contained blocks in Figure 3.

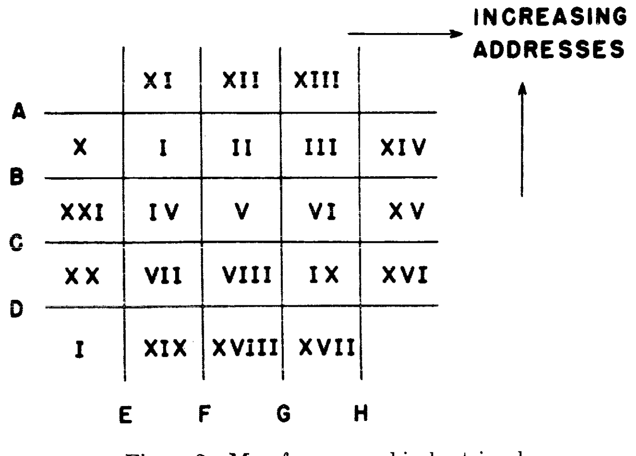


Figure 2—Map for geographical retrieval

As we make the list of contained blocks, we may also make a list of non-contained blocks (Figure 4). These are blocks opposite the contained ones which lie just outside (to the left) of the area boundary. Now we may make a list of blocks adjacent to those blocks listed in Figure 3, excluding blocks already listed as contained or non-contained. This list contains only one block, block V. Enumerating the blocks adjacent to block V we find that all have already been listed as contained. Thus all blocks within the area of interest have been isolated. From the list of blocks in the area, we may develop a list of the address ranges along contained streets or of the parcel numbers of parcels contained in the area.

The algorithm and problem described are reliable only when used with a street network in which no two streets intersect more than once. Techniques have been developed by the author which generalize the algorithm to handle cases in which two streets may intersect more than once, by eliminating resolvable ambiguities or by reporting the presence of irresolvable ones. The generalization requires changing the initial analysis of the list of streets bounding the area from a one-pass to a multiple-pass operation. The first pass isolates all possible sequences of segments which could surround the desired area. The second and succeeding passes eliminate incorrect paths by searching for discontinuities in the transitions from one street to the next. The process is continued until one correct path remains or until no further incorrect ones can be detected.

Two files are used to allow a computer program to implement the algorithm described above. The first contains data about street segments for every street in the map, while the second contains lists of the blocks

I, II, III, VI, IX, VIII, VII, IV

Figure 3—First list of contained blocks

XI, XII, XIII, XIV, XV, XVI

XVII, XVIII, XIX, XX, XXI, X

Figure 4—List of non-contained blocks

adjacent to every block in the map. The first file is used to isolate the sets of blocks just inside and outside an area described by its bounding streets. The segments along a street are ordered by increasing address range, and each segment is described by left and right block numbers, beginning and ending node numbers, and intersecting streets. Additional data on street address ranges and node coordinates for each street are typically included to broaden the utility of the segment file. The block file must include the numbers of the blocks surrounding each block, and should contain data to allow conversion from the numbers of the blocks in the desired area to the data themselves—either as street names and address ranges, as parcel numbers, or as disk identifiers of data records. Both files described above may be produced as by-products of the DIME editing technique[16] described by Cooke and Maxfield.

The algorithm outlined above for using a street network to facilitate geographic aggregation of parcel data has both advantages and disadvantages when compared to the point-in-polygon technique. Its principal advantage is that it is essentially a direct-access technique. The time required to isolate the identifiers of those parcels in an area is proportional to the number and length of the streets surrounding the area and to the number and complexity (number of adjacent blocks) of blocks in the area. Small areas may be isolated very quickly. If some sort of direct-access storage is used for parcel data, the parcels in the area are the only ones retrieved. If sequential storage is used, the algorithm can at least produce a list of parcel identifiers (for example address ranges) which will allow much speedier checking of individual parcels than would be the case with the point-in-polygon routine. The principal disadvantage of the street network technique is its limited flexibility. While the point-in-polygon technique may be used to select parcels in any area, the network technique is clearly applicable only to areas made up of whole blocks. This problem is potentially most serious in analyzing areas such as new highway corridors which do not follow block boundaries. It seems possible that performing such analysis by using the point-in-polygon technique on a set of parcels selected by the network technique might be more economical than applying it to all parcels in a city. However, this hypothesis must be verified.

File structure

The basic implication of the geographic aggregation technique proposed above is that a direct-access file system is very desirable. The principal requirement of this structure is that it be capable of being tied to the block data of the street map file. One flexible way of establishing this tie is to use street address as the major identifier of each parcel and to store street names (or identification numbers) and address ranges in the block file of the street map. The street names and address ranges defining all block faces (one side of a segment) in an area could be merged together and sorted into an order corresponding to that of the parcel data file. Then retrieval from the parcel file could be directed by the sorted output of the aggregation algorithm. Retrieval of data about those parcels in a given area could proceed at a speed governed only by the efficiency of the parcel file's indexing scheme. Variable amounts of data for a single parcel could be stored either in variable-length data records or in multiple files each using street address as primary identifier. Two major advantages of using street address as the primary parcel identifier are, first that all inquiries about parcels by street address would be facilitated and, second, that additions or deletions of occupied addresses within a block face necessitate no alterations to the network data describing that block face.

If a sequential file structure is to be used for parcel data, for reasons of restricted data access, economy, or data volume, the comments about using street address as primary identifier still apply. Although sequential processing becomes imperative, the simplicity of processing allowed by using street address ranges as output from the geographic aggregation algorithm will still minimize the actual processing time required to select parcel data. This minimization may be important when processing data on a small machine or in a partition of a large one.

A planned experimental system

The techniques used above are to be put into practice in an experimental information system for use by the Boston Model Cities Administration and MIT Urban Systems Laboratory. The system will include a street network file and street network geographic aggregation algorithm. The street network file will be tied to a parcel data file by street addresses. Multiple parcel data files will be used to handle multiple data sets (initially housing survey and demographic survey files) on direct-access storage. Control and problem-oriented language facilities will be provided by the ICES system.[17] The system should be implemented by June, 1969 and will be operated as a planning aid for the Model Cities Administration by Model Cities and MIT staff members. In addition to providing basic statistical and cross-tabulation facilities, it is hoped that the system will allow the addition of analytic and modelling capabilities by planning researchers.

ACKNOWLEDGMENT

The work reported here was aided and influenced by many people over the last year. Especially worthy of mention are Professor Charles Miller, Professor Robert Logcher, Mr. William Parsons, Mr. Ronald Walter, Mr. Donald Cooke, and Miss Betsy Schumacker of M.I.T., Mr. Edward Teitcher and Mrs. Colette Goodman of the Boston Redevelopment Authority, and Mr. Michael Warren, Mr. Richard Harris, Mr. Samuel Thompson, and Mr. John Myers of the Boston Model Cities Administration. The work reported herein was conducted at and sponsored in part by the Urban Systems Laboratory of the Massachusetts Institute of Technology.

BIBLIOGRAPHY

1 O E DIAL
Urban information systems: A bibliographic essay
Urban Systems Laboratory M I T 1968

2 S McINTOSH D GRIFFEL
The ADMINS primer
Center for International Studies M I T

3 S McINTOSH D GRIFFEL
The language of ADMINS
Center for International Studies M I T

4 P A CRISMAN
The compatible time-sharing system: A programmer's guide
M I T Press

5 H H COCHRAN
Address matching by computer
Proc Sixth URISA Conference 1968

6 R B DIAL
Street address conversion program
Urban Data Center University of Washington

7 S NORBECK B RYSTEDT
Computer cartography point-in-polygon programs
BIT 7 1967

8 D F COOKE
Systems, geocoding and mapping
Proc Sixth URISA Conference 1968

9 C L MILLER
Man-machine communications in civil engineering
Department of Civil Engineering M I T

10 R E BLEIER
Treating hierarchical data structures in the SDC time-shared data management systems (TDMS)
Proc A C M National Conference 1967

11 E W FRANKS
A data management system for time-shared file processing using a cross-index file and self-defining entries
Proc S J C C 1966

12 K J DUEKER
Spatial data systems
Northwestern University

13 S B LIPNER
File structures for urban information systems
Internal Working Document M I T 1968

14 G L FARNSWORTH
Contiguity analysis using census data
Proc Fifth Annual URISA Conference 1967

15 W A PARSONS
Unpublished class project report
M I T Subject 1 152 1968

16 D F COOKE W H MAXFIELD
The development of a geographic base file and its uses for mapping
Proc Fifth Annual URISA Conference 1967

17 D ROOS
ICES system: General description
Department of Civil Engineering M I T

Automatic traffic signal control systems—the metropolitan Toronto experience

by JOHN D. HODGES, JR.

UNIVAC Division of Sperry Rand Corporation
Los Angeles, California

and

DOUGLAS W. WHITEHEAD

Metropolitan Toronto Roads and Traffic Department
Toronto, Canada

INTRODUCTION

Like a number of fast growing North American cities, Metropolitan Toronto is faced with the ever increasing problems of greater motor vehicle traffic volume, congestion and accidents. Metropolitan Toronto today has more than 700,000 vehicles registered in its 240 square miles, for a per capita density that rates behind only Los Angeles. In addition over 100,000 vehicles from outside the area converge daily on the City.

In 1957, Mr. S. Cass, Metro's Commissioner of Roads and Traffic, along with a consulting firm began to explore the methods open to them to improve existing roads and streets and, in so doing, gaining greater traffic throughput per dollar in comparison with the building of expensive, new main arteries. The concept of computer controlled traffic signals to co-ordinate and thus improve traffic movement seemed to be an answer, providing it was feasible.

Their preliminary investigations indicated that if a general purpose computer were operated in real-time and on-line, it could:

1. Take in traffic information from a large number of vehicle detectors, determine the interval length required at each individual intersection, and optimize these for overall system efficiency considering existing conditions.
2. Determine the optimum time relationship or offset between individual intersection, considering the existing traffic speed and direction of flow.
3. Directly control the individual signals to produce optimum conditions.
4. Check the signal operation and resulting traffic movement to ensure that conditions were optimum.

It was further shown that a real-time computer could alleviate many of the restrictions and problems confronted in using the normal electro-mechanical or more specialized analog equipment to control signals. A computerized traffic signal control system would allow for:

1. Flexibility in changing certain signal phase arrangements and concepts by only changing the computer programs and not the signal equipment.
2. Co-ordination of data between a variety of types and makes of signal equipment and vehicle detectors.
3. Having a variety of operational plans available and implementable in very short periods of time.
4. Collecting current, complete and accurate traffic flow information from all system signals for use in determining system performance, reliability, and optimum traffic control plans.

The computer based traffic control system concept was seen as the best approach to the problem of moving more traffic through Metro Toronto's streets in less time, but some doubts were still raised as to its operational feasibility. So, in the summer of 1959, the Metropolitan Traffic Department began a pilot test of an Automatic Traffic Signal System. Nine traffic signals along 1.7 miles of one of Toronto's busiest streets were linked to a computer and automatically controlled until the Spring of 1961. A comparison of this automatic system with the usual time-cycle control of traffic lights produced these results: in the evening rush hours, computer control reduced the average delay per vehicle by 11 percent; in the morning rush hour by 28 percent; and it reduced congestion by 25 percent. These dramatic results were achieved with a very limited test system and gave credence to the concept of computer controlled traffic signal systems.

In 1961 the Metropolitan Council decided to go ahead with the installation of a Metro-wide automatic traffic signal control system. At that juncture, a time phased implementation program was begun. By the end of 1969, some 850 intersections will be under computer direction. As of February 10, 1969, 594 traffic signals are under the control of a UNIVAC 1107-UNIVAC 418 computer system.

The most important factor in the Metro Toronto system and the one that makes it the most flexible yet attempted, is that every Metro street with signalized intersections is to have sensors reporting to a central computer. Computer decisions are based on traffic situations at different levels. Certain decisions are made at the individual intersection level without regard to anything else. Other decisions are made at the control area level (15 signals), the group level (6-7 control areas) or the system level (5-6 groups). The computer can detect and analyze major traffic movements and make traffic signal adjustments to prevent potential congestion.

As the installation of the system progressed, theories were put into practice; some worked and others fell by the wayside. Through trial and error, two basic modes of control have emerged as practical methods for the system. Due to the flexibility of the system, the computer is able to implement control gradually across the City and combine a simple mode with a more sophisticated and ultimate mode of control.

At the present time there are several different levels of sophistication. Many areas of the City are entirely pre-programmed with all changes in signal timing initiated by time of day or manual intervention. Many other areas have traffic responsive control at critical intersections (TR2 mode) but have area strategy changing by time of day. A few areas have traffic responsive control (TR2) at critical intersections and traffic responsive area control. One area has all intersections traffic responsive and complete traffic responsive area control.

The traffic control system dynamically senses traffic from detectors strategically placed at each intersection. Acting like an inductance loop, they detect the presence of a metal mass and transmit signals to the central computer site via telephone lines. The computer complex consists of a UNIVAC 418 Real-Time Computer that acts as a communications and message switching device interfaced to a UNIVAC 1107 thin-film memory computer and its peripheral equipment. The 418 has controlled 500 signals in a fixed time mode without the 1107.

The actual traffic count is maintained by the computer. By the use of audio tones, signals are sent over telephone lines connected to a Multiplexer at the central site. Through it, the signals are distributed to their correct address in the Input Scanner, which, in turn, is looked at several times per second by the UNIVAC 418. Thus, a detection followed by the absence of a vehicle presence is one car count to be stored in the computer memory. After some data reduction in the 418, the information is transmitted to the 1107, which determines the optimum traffic light pattern for the City and returns the information to the 418.

From there, the information goes to the Output Distributor. The Distributor contains electiical relays that relate to specific signal locations, which open or close in response to the 418. Thereby each unique traffic control box at an intersection can be addressed by the 418 computer.

Monitors in the traffic control boxes provide data to the 418 computer to confirm that the controller responded correctly to instructions. Should any part (or all) of the system malfunction, the computer will relinquish control of the signals effected to local phase timers in the traffic control boxes. To provide complete protections, the 418 must report constantly via a hold relay to each controller; otherwise, the controller will automatically take over on a pre-set time cycle. The complete cycle of examining the traffic situation throughout Metro Toronto, and taking action if required, is performed once every second. At the same time, the 1107 stores data for future analysis and runs other programs concurrently. The benefits derived from such a control system are numerous:

1. The optimization of traffic control signals has

greater throughput and fewer involuntary steps.
2. Flexibility of control allows the system to be tailored to specific areas and situations within the whole system.
3. Manual control of signals by police is reduced, permitting better allocation of police manpower.
4. Accidents and abnormal traffic congestion are sensed and correction methods automatically implemented where possible.
5. Millions of dollars worth of wasted citizens' time and capital equipment can be saved.
6. Reduction in accidents due to better control can save citizens an estimated 1½ million dollars annually, not including doctors' fees, lost time, lawyers' fees and court awards.

The computer system

The computer system consists of a large UNIVAC 1107 general purpose scientific computer with a UNIVAC 418 real-time process control computer as an on-line input-output front end for interface between the 1107 and the traffic signal control and traffic detection equipment.

Primary computer complex

The primary computer, which is used for control and data processing functions, is a UNIVAC 1107 Central Processor equipped with:

1. A thin film control memory having a capacity for 128 words of 36 bits each with an access time of 300 nanoseconds.
2. A ferrite core memory having a capacity for 32,768 words of 36 bits each with an access time of three micro-seconds. This can be enlarged to accommodate 64,000 words, if necessary.
3. A real-time clock providing time resolution to one millisecond and having interrupt capability.
4. Input-output capability for simultaneous transmission of 250,000 words per second over 16 in, or 16 out, channels.
5. Control console featuring a ten-character per second buffered type printer and enter keyboard along with 15 index, 16 arithmetic and 36 control registers.

Peripheral equipment directly associated with the 1107 comprises:

1. A magnetic drum memory having a capacity for 4,700,000 alphanumerical characters with a transfer rate of 360,000 characters per second.
2. Six magnetic tape handlers each providing a recording density of 1,000 bits per inch with an inter-block spacing of three-quarter inches and tape speed of 100 inches per second, with a transfer rate of 180,000 characters per second.
3. A high speed printer capable of providing either single or multiple copies at a rate of 600 lines per minute with 128 characters per line.
4. A card reader with a capacity for 600 cards per minute, each having 80 columns.
5. A card punch capable of producing an output of 150 cards per minute, each having 80 columns.

Secondary computer complex

The secondary computer, which acts as the on-line input-output device for the 1107 to which it is coupled through a special inter-computer synchronizer, is a modified UNIVAC 418 having:

1. A ferrite core memory for 16,384 words of 18 bits each which can be modified to give 8,192 words of 36 bits. In each case, the access time is four microseconds.
2. Input-output capability over eight channels for 18 bit, or four channels for 36 bit words.
3. Control console featuring a ten-character per second buffered type printer and entry keyboard.
4. Input-output facilities using punched paper tape, operating at a rate of 200 input or 110 output characters per second in six columns.

Operation

The basic function of the computer is to inspect each individual signal once per second and to determine if its aspect should be changed. This is done by comparing the elapsed time for the current indication with that time considered necessary to satisfy both the needs of the alternating traffic flows and the system as a whole.

Control plan

To provide for predictable variations in the requirements (both at and between individual signals) and to facilitate the implementation of special arrangements (such as flashing operation) many different control plans are available. They are maintained either in memory or on drum depending on their frequency of use. The plan actually in effect at any time may be changed either manually by console type-in or automatically by time of day and/or volume criteria.

Traffic data file

The computer itself maintains a traffic data file which is continuously up-dated to show in one second increments for each intersection or vehicle detector:

1. The elapsed time for the current interval.
2. The presence or absence of a vehicle on a secondary detection.
3. The occurrence of a pedestrian push button actuation.
4. The number and duration, in 32nd of a second, of the pulses coming from each primary detection.

The 1107 processes the raw data to give traffic volume, speed and density over sampling periods of any desired length. By using appropriate input parameters, estimates of delay, congestion, etc., can also be arrived at; thus, a very complete picture of current traffic conditions can be obtained.

Signal control

The control procedure governing the operation of any signal and repeated continuously at intervals of approximately one second, is as follows:

1. Read-in the Signal Monitor Code.
2. Identify the current interval and note the time for which it has remained unchanged.
3. Check to ensure that the monitor code is valid.
4. If not valid, repeat check in three successive scans. If still not valid, the signal will be dropped from computer control and the operator notified by console print-out that this action has been taken.
5. If the code is valid, check whether or not any pre-emption is allowed or necessary. If so, set the special function selector and terminate the interval.
6. If no pre-emption is required, determine whether the interval may be extended.
7. Determine the required interval length.
8. Issue the change order when the computed and elapsed time equate.

Interval length

For a pre-determined mode of operation, the exact length of all intervals will be specified in the parameter list. However, for traffic responsive operation, only minimum values will be given for those intervals whose duration may vary. The need for extension beyond these minimum values is determined by a special computer sub-routine. A great many different time determination procedures may be used. These may be either for different signals, for the same signal at different times of the day, or for different traffic conditions. For any signal, the actual procedure to be used at any instant will be specified by the control plan in effect. In the general case, variations are only required in the length of the normal green time. But, in certain instances, the advanced green time may also be made responsive to traffic demand.

Where all signals operate independent of each other, the length of the normal green indications can be calculated in any way desired including predetermined, semi-and fully-actuated and fully traffic-responsive. There are very few signals of this nature in Metropolitan Toronto.

Where the operation of signals on a given street or in an area must be co-ordinated to permit progressive movement, the determination of interval lengths must be made in accordance with modified routines designed to accept the limits imposed by the required cycle and offset relationship.

Pickup and dropout

With the computer correctly loaded with appropriate programs, a manual type-in at the 1107 console engages the pickup routine which automatically brings the selected signals under computer control. If the field equipment does not respond to the pickup instructions, a fail message will be printed out and further attempts at activation are made by direct manual instruction from the operator.

Pickup occurs at each intersection as its timing dial reaches the appropriate point. The whole system can be under control in little more than two local cycles which is never more than about three minutes.

A manual type-in terminates system control of signals. A dropout routine is actuated and adjusts the offsets of each group of related signals to a good off-peak compromise which can then be provided by the local timing dial. When the correct relationship has been established, the timing dial is started by the computer and operation continues without interruption. Following dropout, the operation of each signal is monitored for approximately five minutes and any deviation from normal is indicated by a console print-out.

Accomplished in this way, dropout may take up to fifteen minutes to complete, but the signals will hold in the correct relationship for weeks if necessary.

Analysis

The computer can carry out a series of analytic functions on a pre-determined schedule or on demand. These include both on-line and off-line analysis.

On-line analysis

Aspect changes at any one signal or vehicular movement past any one detector can be displayed on the 418 console as they occur. This provides useful information for service personnel engaged in checking equipment. Aspect information is available even when the system is not under computer control.

A RECORD routine analyzes and prints out information from any four signals together with any sixteen detectors to show, at one second intervals, actual clock time, signal aspect, the number of vehicles passed during the previous second, length of the pulse produced by the last vehicle, and the existence of congestion for each approach.

A SENSOR routine is designed primarily for testing the acceptability of detector information. Results are printed out at fifteen minute intervals to provide an immediate record of traffic conditions and an indication of the need for servicing.

Off-line analysis

Various routines are available for off-line analysis of information stored on magnetic tape. These routines can perform the following functions:

1. Print volumes, average pulse lengths, and lane occupancy for individual detectors, or groups of detectors, for any time interval from one second to several hours, and for any combination of days. The information can be presented in numerical or graphical form.
2. Produce summaries of actual aspect timing on a second by second, cycle by cycle or hourly basis. i
3. Draw space tme charts of co-ordinated arteries on a second by second basis.
4. Simulate delays, stops, queue length, etc., for one intersection based on the real vehicular input and actual signal aspects.
5. Calculate congestion for up to 200 approaches individually and as a group.
6. Produce graphs of delay for a variety of offsets and graphs of stops for a variety of offsets, based on an average platoon arrival distribution that is calculated from recorded data. The computer also produces a figure of merit which estimates the importance of co-ordinating a particular link.
7. Produce volume figures which are corrected for double lane counting losses so that computer produced volumes are within ± 10 percent of actual volumes.

After four or five months have elapsed, the detector information contained on the original data tape on a second by second basis is averaged over 15 minute intervals, to provide for long-term data storage and more convenient preparation of weekly, monthly or yearly comparisons. One tape can hold a year of data in this way. This compressed data tape may be analyzed at any time to provide any required information except that concerning signal aspect. The original data tape is normally cleared and re-used after six months, unless some Court action is pending.

Performance

Area control

On almost every street there is, at some point, a natural discontinuity for through traffic. On the other hand, there are many areas in which signals are so closely spaced or traffic conditions are so similar that close co-ordination and a more or less identical mode of operation is mandatory. Combining these two factors, some sixty so-called Control Areas, each of which can be considered as more or less independent unit, have been created. Signals in these Control Areas can be operated in any required way without reference to conditions in adjacent areas. Many of these Control Areas include signals in network formation, only those on a single street, or those on a section of a major arterial route.

The Control Area concept has simplified programming, data handling and evaluation, while increasing operational flexibility. Insofar as is practicable, both the initial connection of signals into the system and all future development work is carried out on a Control Area basis.

Route control

To provide a thorough check on equipment and a later basis for comparison, the initial operation of each group of signals on a common street has been in a strict pretimed progressive mode, using plans prepared by the computer. A comparison between this mode of operation and the previous non-coordinated arrangement shows a very distinct improvement. Over a large area, travel time and the number of involuntary stops decreased by an average of some eleven and forty-five percent respectively. The average speed and number of vehicles passing in a given time increased by some thirteen and ten percent respectively.

During the next stage of development, the same predetermined plans were used. However, critical intersections were put in the TR2 traffic responsive control mode and area control parameters were selected on a volume basis as well as a time of day basis. The thinking here is that any feedback control system must have an inherent lag in response time in order to be stable. Secondly, the time scale on which area control parameters are determined and implemented is quite large. Thirdly, providing the appropriate area strategy, for say the peak of the morning rush hour, creates small problems if done too soon and creates large problems if done too late.

Therefore, if the traffic responsive mechanism has not initiated a particular rush hour plan by a certain time, then that plan will automatically be implemented by time of day.

This type of traffic responsive operation proved quite successful in that the duration of peak hour congestion was considerably reduced, though it was not eliminated, while the increased flexibility allowed the system to automatically adjust for the variations in traffic demand resulting from holidays, sporting events, etc.

In periods of light traffic, one of the major problems lay in the close spacing of many minor signalized intersections which prevented two-way progressive movement at any reasonable speed. To overcome this problem, secondary detectors have been installed and a number of these intersections operated in a semi-traffic responsive mode. Their yield point has been determined by the through street requirements and the minor street use times. Another method used is to have several minor signals operated in a flashing mode, with red presented to the side street and amber to the major. One hundred and fifty signals currently operate in such a way at certain times of the evening.

Critical intersection control

TR2 control

To overcome uneven traffic flow at critical intersections, the proportion of green time allotted to either phase can vary almost directly with instantaneous demand, while retaining a fixed cycle length. It was found that during peak periods the average delay per vehicle could be reduced by some twenty-seven percent.

TR1 control

A very few major arterial intersections are sufficiently isolated that they cannot be considered as part of a progressive system on either street. In these cases best results have been obtained using an improved volume density type of control. In these instances both cycle length and split vary in accordance with the almost instantaneous demand. With this type of operation, it has been possible to reduce the average delay per vehicle to about thirty seconds, or ten percent, while handling a peak volume equivalent to some fourteen hundred vehicles per lane, per hour of green.

With both these modes of control, it has been found that serious trouble could develop if volume alone, or volume and density alone, were used as the basis for split variation. If, for any reason, congestion developed on one approach and not on the others, in a given sampling period the detectors might indicate that the street on which free flow was taking place was carrying the larger volume and hence it would be given the larger share of green time. This process could be cumulative until the congested street is actually receiving its minimum allowable green in spite of its urgent need for more time. To overcome this, a congestion identification routine is used. This routine detects congestion by analyzing pulse lengths and volumes. The routine provides compensation by artificially increasing the count on the affected street, for TR2, and by ensuring a large minimum green time for TR1.

Turning movement control

At a great many intersections where turning movements present a problem but are not sufficient to warrant a completely separate phase, conditions have been greatly improved by using a split phase arrangement. In this arrangement the green for one direction comes on in advance of that for the other. During this usually short interval, the green for the favoured direction is caused to flash rapidly, this alerts the drivers to its presence and duration while, at the same time, allowing the feature to be omitted at any time without the need for special signs. To provide for clearance and increased safety, the flashing green is changed to a steady indication for about two seconds before the opposing direction is allowed to move.

It is hoped that this selection can shortly be made in accordance with traffic demand at least at those intersections where separate turning lanes are provided and detectors can be located to record the movement.

Accident control

A comparison was made of Police statistics for two similar downtown sections of the City, each approximately two square miles in area with some ninety signalized intersections. It was shown that in one where no change was made in signal operation, traffic accidents increased by five and one-half percent over a two year period, while in the system where control was introduced, there was an accident decrease of about seven and one-half percent. A comparison of conditions on three major arteries has shown that accidents have decreased by some sixteen percent although the volume of traffic has increased by some twelve percent.

Snow plans

It has been found that during even light snow, control efficiency drops sharply simply because of the time required to start a vehicle on slippery pavement. (This is especially true for any approaches on inclines.) On the theory that once movement has started it should be allowed to continue for as long as is reasonable, special plans have been designed to provide considerably longer than normal signal intervals. Longer ambers are also used with these plans.

Economic benefits

Given the amount of accident reduction that the computerized traffic control system has produced (some 10 to 15 percent), it can be safely projected that the system should be saving the citizens of Toronto and its environs some 1½ million dollars annually. This is based on the fact that property damage from auto accidents in Toronto runs to well over 15 million dollars annually. The savings from the increase in traffic flow has been estimated at about 18 million dollars per annum, based just on vehicle operating costs. If the personal time saved could be estimated in dollars, this figure would increase dramatically. Thus, we have a community savings of some 20 milllion dollars per year on a capital outlay of only five million dollars for the total system.

When the computer was moved recently from the old City Hall to the new Police Headquarters, Toronto residents found themselves caught up in a three-week traffic jam since the lights were on their own automatic controls. Motorists were warned of the move and cautioned to leave for work earlier than usual. Once the Computer System, that has reduced traffic tie-ups by 20 percent was no longer controlling the traffic lights, motorists found that in some cases they needed up to an extra hour to get to work. The earlier system predictions of the saving of some 9,000 vehicles hours of delay per day proved to be a gross understatement during those hectic weeks.

ACKNOWLEDGMENT

Much of this report is based upon the efforts and documentation of J. T. Hewton, Metro-Toronto Traffic Control Center, Senior Operations Engineer from 1963 to January 1968.

REFERENCES

1 A CHRISTENSEN R B CODY
Methods of traffic signal control with an electron computer
Traffic Research Corporation
2 KCS LIMITED
A centrally-controlled traffic signal system for metropolitan Toronto
3 TRAFFIC RESEARCH CORPORATION
The control of traffic signals in metropolitan Toronto with an electronic computer
4 D WHITEHEAD
The Toronto system: Intersection evaluation and control
Metropolitan Toronto Roads and Traffic Department

A panel session—Education of computer professional

ELLIOTT I. ORGANICK, *Chairman of Session*

University of Houston; currently at
Massachusetts Institute of Technology
Cambridge, Massachusetts

Inter-relating hardware and software in computer science education

by JACK B. DENNIS

Massachusetts Institute of Technology
Cambridge, Massachusetts

The major portion of graduates from curricula in computer science will be professionally involved in the design, specification, implementation or theoretical foundations of computer-based information systems. They will participate in the selection of computer hardware, or will be called on to judge the merits of proposals from suppliers. To be competent in exercising these responsibilities, it is essential that students of computer science thoroughly understand the relationship between computer organization and the implementation of programming languages and information systems.

Given this objective, there is a serious anachronism in the teaching of programming and computer organization in contemporary university curricula: Computer organization is often taught as the final example in a course on logical design by instructors who do not profess knowledge of compiling techniques and software issues. Conversely, programming courses are based on conventional assumptions of computer organization (Von Neuman) as if they were axioms of nature. Moreover, the communication paths that could lead to reorganization and accommodation of the intellectual substance of both areas are frequently blocked by circumstances: Either the areas are the "property" of separate academic departments, or the faculty is divided by disparate interests.

There is critical need for cooperation between faculty in programming and in computer organization to jointly develop curricula that interrelate hardware and software principles for realizing the functional requirements of computer systems. The ACM Curriculum Proposal does not represent sufficient progress toward this objective.

In the undergraduate Computer Science program of the M.I.T. Department of Electrical Engineering, we have developed a three subject sequence in computer systems and programming intended to interrelate software and hardware principles:

1. Programming Linguistics
2. Computation Structures
3. Information Systems

Students enrolling in the sequence are presumed to have had the experience of expressing programs in an algebraic language and seeing them run (with success and failure) at a computer installation.

The first subject, "Programming Linguistics," treats the important concepts in describing and interpreting algorithmic procedures on the basis of a formal semantic theory. Features of practical programming languages are related to the theory. Discussion of hardware is deliberately omitted so an unencumbered appreciation of linguistic principles can be achieved.

In "Computation Structures" the student learns the properties of memory and logic components that interact strongly with the process of planning a computer organization. A graph model of parallel computations is used both to describe modular hardware systems, and, as a starting point for developing combined hardware and programming techniques, for realizing the linguistic features studied in the first course.

The subject "Information Systems" makes use of the material from the preceding subjects in studying the analysis, design and implementation of computer-based informations systems.

The second subject of this sequence is the key to inter-relating hardware and software concepts. A significant difference from conventional curricula is that students begin in their study of "Computation Structures" with a thorough understanding of the features found in a variety of source programming languages. With this background they are prepared to study the principles by which these features may be made available to computer users through the combination of hardware technology and programming concepts. An outline of this subject as it is currently taught at M.I.T. is given below.

Computation structures

1. Logic Design: Elementary combinational circuit synthesis; registers and gating; asynchronous modular systems (macromodules); sequential circuit synthesis; elementary implementation of arithmetic operations.
2. Memory Systems: Physical principles; name space—value space; distinction between location-addressed and associative memories; addressing by key transformation for associative retrieval.
3. Computation Schemata: Representation of a computation (in digital logic or as an abstract algorithm) by a set of *operators* that transform the contents of a set of memory cells. The domain and range cells of the operators are indicated by a *data flow* graph. The constraints that govern the sequencing of operator applications are specified by a *precedence graph.* Necessary and sufficient conditions for deterministic (unambiguous) operation are formulated. Extensions are made to represent procedures involving decision and iteration.
4. Machine Organization: Study of the principal forms of single-sequence processor organization and the corresponding techniques for compiling arithmetic assignments and conditional expressions: A simple single address machine; a stack-organized machine; machines having multiple general registers; machines having several functional units.
5. Parallel Processing: Multiprocess computer systems; process state, supervisor programs and scheduling; primitive procedure steps for representation of parallel computations: The *fork, join* and *quit* primitives; Dijkstra semaphores—the P(s) and V(s) operations. Process interlocking problems and their resolution.
6. Nesting and Recursion: Representation of an operator of one schema by a second schema. Naming of input and output quantities. Occurrence of multiple activations of a procedure through parallelism or recursion. Local data areas; use of stack storage allocation for single process implementation; base registers.
7. Information Structures: An information structure is modelled as a directed graph, without directed cycles, containing a directed path (not necessarily unique) to each node from a particular node called the *root.* Static operations on information structures; implementation by use of indexing. Dynamic operations; implementation by linked blocks in a location-addressed memory; garbage collection; implementation with associative memory.

Let's not discriminate against good work in design or experimentation

by GEORGE E. FORSYTHE

Stanford University
Stanford, California

I am distressed that graduate education in computer science is forcing students into a theoretical mold, and away from the vital practical problems of software engineering. I therefore urge that graduate computer science departments pay attention to the problems of experimentation and design in computer science. This might be done, for example, by employing faculty with interests in design and experimentation, by offering courses and examinations in these areas, and/or by accepting Ph.D. dissertations involving substantial designs or experiments of high quality. I believe that the last is the most important action to be taken now.

It seems to me that one main function of an educational system is to furnish society with imaginative performers and potential leaders in all the various areas of life. Within the computing field, there is a huge need for persons to create well designed, well documented software systems that exploit computers in the manifold ways we know to be possible. While the field will surely be advanced farthest by the creation of good new ideas, there remain enormous steps to be taken in exploiting ideas already known. Thus our educational system in computing should do three things:

1. teach the leading ideas now believed to be relevant to computing, and inspire students with the desire to keep on learning after they graduate;
2. seek out and inspire a few leading minds capable of augmenting our stock of ideas with good new ones and show them how to do it;
3. inspire a generation of students to design and experiment with good systems with the methods now known and soon to be discovered, and show these students how to do a good job of it.

By accepted custom, the Ph.D. degree requires the student to perform in steps (1) and (2): he passes examinations in relevant ideas, and he writes a dissertation whose main requirement is to contain some original theoretical work. I believe these two steps are entirely correct for a Ph.D. in a theoretical subject like pure mathematics.

But the very point of founding schools of engineering and departments of computer science was that society needs concentration on work relevant to today's technology. This implies a certain abandonment of learning for learning's sake, in favor of work on problems whose solution is actually needed. In the computing field, this implies to me that we must not confine our students to Ph.D. dissertations that are of the classical type, but should be prepared also to accept first class work in design or experimentation.

Students are attracted to computer science because it has a lot of action, rather than just contemplation. From the start, our students are creating programs that *do* things, and they enjoy it. Many of them are eager to keep on designing and programming systems, and it seems almost criminal to turn this eagerness off. Instead, our graduate departments should be accepting this urge to produce, and concentrating on channeling the design, experimentation, and production into worthwhile projects, done with high standards.

It has been argued that design and experimental work are fine, but should be rewarded with a different degree—one analogous to the degrees of Engineer or Doctor of Arts. I disagree, mainly because different degrees tend to acquire different levels in the hierarchy of snobbery, and I refuse to admit that excellent work in design is any less important than excellent work in theory. The Ph.D. degree has become the accepted reward for first class performance in graduate school (e.g., in experimental physics), and should be retained in that function. Any further assessment of the quality of a person's work can be passed along in personal letters of recommendation. If my recommendations were followed, I would expect to see more Ph.D. theses with titles like

"A very high-performance compiler for PL/1 on System/360";

"Study of all calls on the scientific subroutines on the CDC 6600 at NYU in October 1969, and a resulting proposal for reorganizing the library."

In summary, the purpose of our educational establishment is to reward students for developing their educational and performance potential as much as they can. Let us use the Ph.D. as a reward for first class work in *any* aspect of our field, and not discriminate against work in design or experimentation.

Applied computer science

by LOTFI ZADEH

University of California
Berkeley, California

It is a truism that we are in the throes of an information revolution of which one obvious manifestation is a very rapid growth in the number of users of computers and computer-like information processing systems.

It is also evident that the number of computer users is growing much more rapidly than the number of computer scientists and engineers. As a result, computer science may become user-influenced to a much greater extent than other fields of science and engineering. This was in evidence at the 1968 IFIP Congress in Edinburgh, at which the ratio of non-professional users to computer scientists was far greater than, say, the ratio of non-mathematicians to mathematicians at the 1966 International Mathematical Congress in Moscow.

The overwhelming preponderance of computer users over computer specialists is certain to have a profound impact on computer science education in the years ahead. One likely effect is that much of the training in the use of computers will be taking place outside of computer science departments and will be tailored to the needs of students in particular fields. Another possible effect is a splitting of computer science into pure computer science and "applied" computer science a la the division of mathematics into its pure and applied branches.

If one believes, as this panelist does, that an organizational fractionation of computer science into "pure" and "applied" components is undesirable, then greater attention will have to be devoted to making it possible for a computer professional to receive a Ph.D. or equivalent degree on the basis of a thesis that is primarily applied in nature. By this is meant a thesis which contributes not to computer science per se but to a nontrivial application of it in some field external to computer science. For example, an acceptable Ph.D. thesis of this nature might deal with the application of computers to medical diagnosis; or to problems in air traffic control; or to simulation of neural nets, etc.

It is essential that a student electing to do his dissertation in an applied field should devote a substantial amount of time to familiarizing himself with it. Thus, if his thesis is concerned with, say, the medical diagnosis by computer, he should be prepared to spend a month or two in a hospital acquainting himself with various aspects of medical diagnostics. The Ph.D. dissertation of G. A. Gorry* of Project MAC, M.I.T., is an excellent example of a thesis of this type.

In summary, although one can find isolated examples of Ph.D. theses in what might be called "applied" computer science in various institutions, this panelist believes that a conscious effort should be made to encourage work of this type within computer science and electrical engineering departments and accord it the same respectability as research in pure computer science.

Identifying and developing curricula in software engineering

by ALAN J. PERLIS

Carnegie-Mellon University
Pittsburgh, Pennsylvania

One basis for developing an education program is the recognition of a continuing need for a certain class of professionals in our society. The need may be redressed because:

An influential or significant part of the society may have a need for professionals that is not being met by the educational system.

An influential or significant part within the education system may observe an unrecognized need of society and begin to prepare what will eventually be needed.

* G. A. Gorry, "A System for Computer-Aided Diagnosis," September 1967, MAC-TR-44.

The educational system may not respond to external pressures because it sees the need as temporary or non-critical, or it just may not be interested in such problems. The educational system may say it hasn't the resources to provide the professionals or settle for a merely adequate solution. It may even delay solving the problem by an act of generalization or systematization that, at best, functions to postpone.

The present graduate programs of computer science arose from the second process. As a result there is an aspect of computer education that is not being provided in response to a situation of the first kind. I refer of course to the education of a class of people that I shall call software engineers whose training and goals are quite different from those of our current computer scientists.

These badly needed people are engineers and their domain of specialization is software—its design, production, and servicing, Actually this is too restricted a view: Their specialization is computerware, both hard and soft. Actually they deal with a spectrum of states of computer matter each stable only in certain environments.

It requires little insight or sleuthing to see that such engineers are in very short supply and the need for them is acute. Accepting this state of affairs, certain questions need to be put and then to be answered by the educators and the employers of such people.

1. Should training be conducted in the university or in technical or trade or junior college schools or indeed in all?
2. If in a university, in what department or discipline or school should it be conducted?
3. If in a department or interdisciplinary program, at what levels should degree programs be offered and in what order of priority?
4. Do these programs exist in the steady state as a separate discipline or as a minor or option or modification of existing ones?
5. Is the need going to persist and, if so, will it persist in its present form? Will the people be educated for the coming generations of systems problems?
6. Why do we speak of engineering and not science?

It is here suggested that the proper place to start is at the master's level. From that program one can move up to the doctoral and down to the bachelor's programs. The master's program should not be a faceless one turning out computer scientists whose grasp and reach could not attain the computer science doctoral degree. The engineering component of this education is paramount. The goal should be concentration on known tools and their effective use and not on periods of intense innovation or discovery. The choice from competing designs is more important than the discovery of the existence of designs. The issue of stability is more critical than that of growth and change. The determination of task magnitude is as important as the discovery of method. The directing of teams is as critical as the spark of breakthrough. The professional accomplishment of mean tasks that are of peripheral importance to the society are distinct from the devotion only to the bizarre, rare and new.

At the moment there are few studies being made of the problems of training these people. Rather than reproduce a curriculum here, I should like to list questions that a trained engineer should be able to answer:

1. Given a software task, similar to familiar ones, and a set of computers, evaluate the machines and the task to make a meaningful choice of machine and representation of the software system that is optimal. Ho! What is optimal? How relevant are the issues of manufacturer support? Compatability? Stability? Natural gradients for change, growth, and improvement?
2. Given a software task, what is a rational schedule for its completion given various personnel situations? How do you get, train, or even recognize adequate programmers? How do you set up work loads?
3. If n people are programming a system, what do you do when the n + 1st arrives, etc.?
4. How do you test a system? What kind of a system do you organize to handle and respond to pressures on a system?
5. How do you market a system? What makes a system useful? How do you copy someone else's system?
6. How and what do you learn from building a system? What goes into the inventory stockpile after spending time on a software task?
7. What are the tools of the trade? How are they catalogued? Related to diverse hardware?
8. How do you tie together disjointed systems into coupled ones solving enlarged tasks?

I look upon the professional education of software engineers as an amalgam of mathematics, management science, computer science and practical experience gained from contact with actual software systems and associated problems.

Some of the master's may continue onto the doctorate in computer science, but the program is not seen as being merely preparation for a doctorate in computer science.

SAL—Systems Assembly Languages

by CHARLES A. LANG

University Mathematical Laboratory
Cambridge, England

INTRODUCTION

The Cambridge Computer-Aided Design group is writing some general purpose software tools that aim to assist scientists and engineers to apply their problems to the computer with maximum ease. These tools include a storage allocation system, a data structure package, a compiler-compiler for mixed graphical/verbal on-line languages, a package of procedures for generating pictures and transmitting them to a display, plotter, or file, and programs for operating a link between a multi-access computer and a satellite computer. When the group started late in 1965 it had to determine what language to use to write these systems. After struggling with the difficulties of assembly code for some time for those programs for which FORTRAN was unsuitable, we decided to design and implement a more suitable language; Systems Assembly Language (SAL) is the result. The purpose of this article is to explain the thinking behind SAL rather than to expound on the finer details of the language itself. We feel that this type of language which combines the freedom and flexibility of assembly code with many of the facilities normally associated with high level languages, could be useful to many other workers. Further, this type of language could perhaps usefully be provided on all computers.

Design requirements for SAL

The design of a programming language is a compromise between various requirements; the requirements for SAL are now discussed.

Clarity

A programming language provides the conceptual framework within which a programmer must think about his problem, so influencing his method of solution. The language should enable the programmer to state clearly the computing operations he wants performed; further, the program must be clear to read both by himself and others. A high level language such as FORTRAN, ALGOL or AED meets these requirements better than assembly code, but these languages fail to meet the other requirements discussed below.

Effectiveness

The language itself must be easy to learn (a short programming manual is desirable), and capable of being compiled efficiently to produce efficient code. These requirements must not be achieved at the expense of the language's generality or practicability. The target is a high "power to complexity" ratio, the view being taken that if the basic operations provided by the language are well chosen, then these provide the tools with which the programmer may fashion more complicated and specialized operations. We have tried to achieve this by making the syntax of SAL very general, avoiding exceptions and special cases as far as possible. Also as much as possible has been left out of the language rather than the reverse. For example, a stack is frequently used in system programs, so at first we planned facilities to set up, operate and destroy stacks. Later, however, we abandoned this idea as we realized that the kind of stack we were planning (a first-in, first-out stack which stacked single computer words), might not be the type of stack required by all. Furthermore, such a stack could very readily be programmed using other features of the language. To build in the generality for all possible forms of stack would have been too complicated for this type of language. We have met our target of producing the compiler, documenting it and writing a programming manual[1] in a single man-year. None of the contending languages available in our laboratory, FORTRAN, ALGOL, and CPL meet the above requirements; they were, of course, never intended for use primarily as languages for writing system programs.

There exists a whole spectrum of languages starting with assembly code at the lower end, then higher level machine dependent languages, then higher level machine independent languages, and eventually reaching the highest level ones such as PAL[2]. To put SAL in perspective it comes somewhere inbetween a simple machine independent language such as BCPL[3], and a machine dependent language like PL360[4], both of these languages being intended for writing compilers and systems. In PL360 assembly code type instructions are written together with ALGOL-like declarations, iterative and conditional statements and a block structure. SAL contains several high level facilities not available in PL360, so is roughly parallel in the spectrum with MOL940,[5] a language for the SDS 940 which "permits the expression of clear, concise algorithms and the production of efficient tight code".

Suitability for writing system programs and machine dependence

The language must contain basic features required for writing system programs. In particular, these include explicit address manipulation, data structures, control of iterative and conditional operations, bit manipulations, character manipulations, arithmetic and logical operations, intercommunication between separate program modules (perhaps written in different languages), control of peripherals, and the ability to separate program and data (so that a single copy of a program may be shared in a multi-programmed environment). SAL attempts to provide these facilities as described in a later section, though it is not currently strong on character manipulations.

While we applaud machine independent programming wherever possible, we recognize that there are times, even if very few, when machine dependent programming is desirable, for example when very high efficiency is essential, or intimate control is required over the machine such as in the control of peripherals, or loading of machine registers. SAL caters for this machine dependence in two ways. First, machine registers may be referred to in any arithmetic, logical or conditional expression. Second, assembly code may be embedded quite freely anywhere in a SAL program and may refer directly to declared variables. This permits easy communication between the assembly code and the rest of the program. This feature was specified in the ALGOL 60 report[6] and has been included in many implementations of ALGOL and FORTRAN and also in AED.[7] Both PL360 and MOL940 include both of these features.

No run-time system

The language must have no run-time system of any kind. That is, nothing shall be loaded along with the program "behind the users' back", such as an input/output package, or run-time routines to operate stacks or dynamic use of working space. In this sense, SAL is like a conventional assembly language. This requirement allows SAL to be used to write any type of system, leaving the programmer quite free to control everything loaded into the core along with his system program. Further, the consequence of each statement should be clear to the programmer.

Ability to run together programs written in different languages

The general purpose system programs we write are intended to be used as tools by other programmers for particular applications. Despite the many attempts, no programming language has yet been devised which is suitable for all programming tasks. Apart from this the more comprehensive a language becomes the more unwieldy the compiler and system that goes along with it tends to become also. The user of the tools must not be forced to write his own application programs in the same language as used to write the tools. Rather, he should be free to write his programs in whatever language is most suitable for his application. This means that SAL programs must run within some mixed language system where programs written in separate languages may be compiled separately but loaded and run together. Many such systems exist, but often the methods used for interlanguage communication (which are a function of the loading system as well as the languages) are limited to the FORTRAN requirement. This provides one way of calling procedures and passing arguments, plus communication via "COMMON." As a result of these restrictions, some languages, notably ALGOL, are not included in a mixed language system. The same loader, however, is sometimes used to load into core the compiled code of these languages, and of languages within such a mixed language system. A further requirement is that a SAL program must have equal freedom to assembly code programs, to communicate with programs written in any language that may be loaded into core with a SAL program by the same loader.

Examples of mixed language systems are the Project MAC system on the IBM 7094 where assembly code, FORTRAN, MAD, AED, etc., programs may be loaded together, or the SCOPE system on the CDC 6600. The system at Cambridge is known as The Mixed Language System (MLS) and will be referred to later.

The language

The language description is divided into two sections. First there is an example, rich with comments, that uses many features of the language. Second, the unusual or special features of the language are described. The syntax is listed in the appendix. SAL is designed for the Titan (Atlas II) computer, which has a 48-bit word. Floating point operations use the single 48-bit accumulator. There are 128 index registers (called B-lines) of 24 bits each. All integer arithmetic is performed on half-words (24 bits) using the index registers as accumulators.

Examples of SAL programs

The following examples show what a SAL program looks like. An attempt has been made to use as many of the facilities of the language as possible rather than to write the most succinct program. These examples, in conjunction with the syntax of SAL in the appendix, attempt to convey concisely the facilities of SAL. Only the more unusual facilities are described in more detail in the sections below.

Program 1 generates a picture of a cubic and transmits it to a satellite computer with a display. Program 2 is a group of procedures which may be used to generate a picture consisting of lines and points for the Digital Equipment Corporation PDP7/340. Both programs are highly annotated, comments being introduced by a bar (|) and terminated by a bar, semi-colon or newline. They have been tested and proved to work. It is suggested that readers take a first quick look at the program example here. They should not, however, expect to understand the details until they have finished reading this paper.

PROGRAM 1

```
|main program to plot a cubic on the display

GLOBAL LABEL BUFFER,POINT,LINE,SEND |entry points to procedures
                                    |in program 2
GLOBAL LABEL START |entry point to this main program
INTEGER X,XTHIS,YTHIS,XLAST,YLAST,WSPACE(15),DFILE(100)
BLINE 2 ARG1,ARG2,ARG3 90 LINK
|set up variable space for pure procedures
                                    GLOBAL LABEL DUMP
START,   B77=PTR WSPACE(0)
         OSELECT 0          |output stream selected for messages
         X=1
|set up buffer for display file
         ARG1=PTR DFILE(0); ARG2=100; ARG3=DFERROR;
         LINK=MAIN1; GOTO BUFFER;
MAIN1,   |plot a point at 0,4| ARG1=0; ARG2=4;
         LINK=MAIN2; GOTO POINT
MAIN2,   XLAST=0; YLAST=4
LOOP,    WHILE X<11 DO
                XTHIS=X*10; YTHIS=X*(X*(X-2)+3)+4
                ARG1=XTHIS-XLAST; ARG2=YTHIS-YLAST
                X=X+1
                XLAST=XTHIS; YLAST=YTHIS
                LINK=LOOP
                GOTO LINE
         REPEAT
|send display file to the satellite
         LINK=DONE; GOTO SEND
DONE,    OSTRING ::DISPLAY FILE TRANSMITTED
::;      STOP
DFERROR,        OSTRING ::DISPLAY FILE OVERFLOW
::;      STOP
         FINISH
```

PROGRAM 2

```
|display file system
|this routine contains procedures buffer,point,line,send
|the procedures are written as pure procedures, so use external
|variables rather than global, local, or common variables
|arguments are passed to the procedures by value in index registers
|B2,B3......etc, and the result, if any, is returned in B1.
|on entry to the procedure the return address is in B90

GLOBAL LABEL BUFFER,POINT,LINE,SEND |entry points to procedures
RELSTART INTEGER B77 0
INTEGER B77 DFLOC,DFOVER,DFNEXT,DFEND,MODE,NEXTMOD,TEMP
BLINE 2 ARG1,ARG2,ARG3 90 LINK  |mnemonic names for index registers

|set up manifest constants
        PARWD IS O14117; XCOM IS O22000; YCOM IS O222000
        DXYCOM IS O2000000; ENDCOM IS O402000; HALF IS O4

|buffer(dfloc,dfsize,dfover)
|procedure buffer is called to set up a buffer at address dfloc
|and size dfsize in which the display file is to be built, one word
|of display file per half word. if it overflows exit from buffer
|is made to address dfover


BUFFER,         DFLOC=ARG1      |pick up first argument
        DFEND=ARG2+ARG1 |dfend marks end of display file buffer
        DFOVER=ARG3
        →DFLOC=PARWD    |first word in the display file is parameter
            |mode, setting scale,intensity and light pen sensitivity
        MODE=PAR        |mode of last word in file is parameter
        DFNEXT=DFLOC+HALF |dfnext marks the next free buffer location
        GOTO LINK       | return to calling program



|point(x,y)
|procedure point adds two commands to the display file to set x and y
|x and y must be positive and less then 1024

INTEGER B77 COORD,RETURN
POINT,  |set dispaly file to receive point mode words
        RETURN=POINT1;  NEXTMOD=PO;
        GOTO MODE
POINT1,  |check to see if there is space in the display file buffer
        IF DFNEXT+1 GE DFEND THEN GOTO DFOVER END
        COORD=ARG1 AND 1023
        COORD=COORD SHIFTRC 3
        →DFNEXT=XCOM+COORD       |x point command
        DFNEXT=DFNEXT+HALF
        COORD=ARG2 AND 1023
        COORD=COORD SHIFTRC 3
        →DFNEXT=YCOM+COORD       |y point command
        DFNEXT=DFNEXT+HALF
        GOTO LINK       |return
```

```
|line(delx,dely)
|line adds words to the display file to generate lines.
|delx and dely may be in the range -1024<DEL<1024
|the maximum length that may be displayed by one command
|is 127, so several commands may have to be built


INTEGER B77 N,DELX,DELY,INCRX,INCRY,REMX,REMY
LINE,   IF ARG1 GE 0 THEN DELX=ARG1 ELSE DELX=-ARG1 END
        IF ARG2 GE 0 THEN DELY=ARG2 ELSE DELY=-ARG2 END
        IF DELX GE DELY THEN TEMP=DELX ELSE TEMP=DELY END
        N=1
        WHILE TEMP>127*N DO N=N+1; REPEAT
        INCRX=DELX/N;   INCRY=DELY/N
        REMX=DELX-INCRX*N;      REMY=DELY-INCRY*N

|build display file commands
        IF DFNEXT+N/2   |N/2 as each command occupies a half word|
        GE DFEND THEN GOTO DFOVER END
        RETURN=LINE1;   NEXTMOD=VEC;    GOTO MODE
LINE1,  FOR N=N STEP -1 UNTIL 1 DO
                DELX=INCRX
                IF REMX GT 0 THEN REMX=REMX-1; DELX=DELX+1 END
                IF ARG1 LT 0 THEN DELX=DELX AND O2000 END |neg bit
                DELY=INCRY
                IF REMY GT 0 THEN REMY=REMY-1; DELY=DELY+1 END
                IF ARG2 LT 0 THEN DELY=DELY AND O2000 END |neg bit
                DELY=DELY SHIFTLC 8
                →DFNEXT=DXYCOM+DELX+DELY
                →DFNEXT=→DFNEXT SHIFTRC 3
                DFNEXT=DFNEXT+HALF
        REPEAT
        GOTO LINK
|end of procedure line


|procedure send transmits the display file to the satellite
SEND,   |close off display file
        IF DFNEXT+1 GE DFEND THEN GOTO DFOVER END
        RETURN=SEND1; NEXTMOD=SUB; GOTO MODE
SEND1,  →DFNEXT=ENDCOM; DFNEXT=DFNEXT+HALF
        B10=DFNEXT-DFLOC            |number oft words
        B11=2000                    |address in satellite
        B12=0                       |Data word for satellite
        B13=DFLOC
        1011    0       0       0       |select link to satellite
        1013    10      13      0       |send display file
        GOTO LINK


        ICOMP MOD 0,O160000,10
|parameter word
INTEGER B77 MODNUM
PAR,    IF NEXTMOD=PAR THEN GOTO RETURN END
|set up mode change in previous word
PAR1,   IF NEXTMOD=PO THEN MODNUM=1 ELSE
                IF NEXTMOD=VEC THEN MODNUM=4 ELSE MODNUM=7 END END
        MOD(DFNEXT-HALF)=MODNUM
        MODE=NEXTMOD; GOTO RETURN
```

```
|point mode
PO,      IF NEXTMOD=PO THEN GOTO RETURN END ;    GOTO PAR1

|vector mode
|vector mode words are treated differently. They must
|first escape into parameter mode by addition of a single bit,
|then a whole new parameter word with the required mode change
| must be added to the display file

VEC,     IF NEXTMOD=VEC THEN GOTO RETURN END
         →(DFNEXT-HALF)=→(DFNEXT-HALF) OR 0400000 |add escape bit
         IF DFNEXT EQ DFEND THEN GOTO DFOVER END
         →DFNEXT=0          |set null parameter word
         DFNEXT=DFNEXT+HALF
         GOTO PAR1

|subroutine mode
SUB,     GOTO PAR1

FINISH
```

Special features of the language

Sections of program ending with the word FINISH are known as *routines*. A routine may itself contain several procedures. Note that statements may be terminated either by a semi-colon or by a newline.

Declarations, control of space, and intercommunication between programs

Particular attention has been given to the ability to write pure procedures, i.e., separation of code and data, control of data space, and communication between programs, whatever language they may be written in. Variables may be declared in different data spaces as explained below. Most operations in SAL are carried out with 24-bit words, called half-words in Titan terminology. Variables may be declared either as type INTEGER (for want of a better name) or as type LABEL. This paucity of types simplifies the compilation process, and gives the programmer more flexibility, as he can assign any conceptual type (integer number, logical, string, character, list head etc.) to an integer variable and mix these conceptual types in arrays and other forms of data structure. Integer variables may be declared in several ways as discussed below. A special feature of SAL is that labels may be mixed with integer variables in arithmetic and logical expressions (see next section).

No floating point operations are provided in SAL except by using embedded assembly code. REAL variables may be declared (distinct from integer variables as they are 48-bit numbers) for use with this embedded assembly code as well as for communicating with FORTRAN programs.

Local Integer Variables

e.g., INTEGER X, XTHIS, YTHIS

These variables are local to the routine in which they are declared. Space is assigned for them by the compiler at the end of the code of the routine. Consequently, the local integer variables of separate routines do not share the same space in core.

Global Variables

e.g., GLOBAL INTEGER RED, WHITE, BLUE

Global integer variables allow for communication between separate routines.

e.g., if routine A declared:

GLOBAL INTEGER RED, WHITE, BLUE

and routine B declared:

GLOBAL INTEGER GREEN, RED

then both routines may reference the same variable RED. Space for global integer variables is allocated by the loader at load time.

External Integer Variables

External integer variables provid three facilities: the ability to write re-entrant code, dynamic allocation of space, and a further method of communication between routines. These variables are not referenced directly, but *relative* to an address held in an index register. It is the programmer's responsibility to store in an index register the address of some space that is to be used for the variables.

three of which are:

INTEGER B77 DFLOC, DFOVER, DFNEXT

Before such a declaration is made, however, the user Program 2, for example, uses index register 77, and declares several external integer variables, the first must declare where, relative to the address in the index register the variables are to be allocated:

e.g., RELSTART INTEGER B77 0

This allows variables for different routines, referenced relative to the address in the same index register to be separated. For example, another routine might make the declaration:

RELSTART INTEGER B77 7

The space used by the external integer variables in Program 2 is set up in Program 1 by the statement:

B77 = PTR WSPACE(0) (PTR is explained in next section)

Now let us consider the three facilities mentioned above, starting with re-entrant code. As external integer variables are referenced relative to an address in an index register, a new set of variables may be referenced by changing this address. The Titan operating system is multi-programmed, and saves and restores index registers for each program when switching between programs. Hence, several users may share the same program, each one setting up the same index register to contain the address of space to be used by the variables. Single users too may have requirements for re-entrant code to implement recursive procedures, or to handle program traps (for example from the satellite computer). In this single user case, it is the user's responsibility to ensure that the address in the index register is set up before re-entering some code.

External integer variables also provide a method, under user control, to dynamically allocate space, producing an effect similar to that provided automatically by some block structured languages. Suppose, for example, there was declared in one section of code:

RELSTART INTEGER B25 0

INTEGER B25 P,Q,R,S

and in another:

RELSTART INTEGER B25 0

INTEGER B25 OAK, ASH, ELM

then assuming that the content of index register 25 was identical in each section of program (the address of an area in core for use by these variables) then P,Q,R would use identical locations to OAK, ASH, ELM.

Finally, routines may communicate via external integer variables. For example, if two routines both declared:

RELSTART INTEGER B25 0

INTEGER B25 P,Q,R,S

then P,Q,R,S would be accessible to both routines. Index register 25 could be set up by either of these routines, or another one.

Index Registers

Index registers may be referred to by giving their number prefixed by B, for example, B2, B3, B4, B90, or they may be referred to symbolically by making a declaration:

e.g., BLINE 2 ARG1, ARG2, ARG3 90 LINK

This declaration makes ARG1, ARG2, ARG3 equivalent to B2, B3, B4 and LINK equivalent to B90.

Common Variables

Common variables are included in SAL purely to permit communication with routines written in FORTRAN:

e.g., COMMON /SHAPES/ ROUND, SQUARE

The name between the / / enables named common to be referred to for communication with FORTRAN IV programs.

Arrays

Single dimensional arrays are declared by giving the size of the array after the name:

e.g., INTEGER YLAST, WSPACE(05), DFILE(100)

INTEGER B27 A, B(50), C(50)

GLOBAL INTEGER D,E,F,G(1000)

COMMON /SHAPE/ PENTAGON(5)

Space for arrays is allocated in the same way as for single variables. Local integer arrays are assigned by the compiler at the end of the routine, the user must set up the space for external integer arrays, and the loader assigns the space for global integer arrays, as it does for common integer arrays which are actually assigned within the common area.

Reals

Real variables may be declared for use with embedded machine code, and as arguments when calling FORTRAN procedures (see section on procedures), by the declaration:

e.g., REAL NUMBER, RESULT

Labels

A local label is declared automatically by placing one in a program:

e.g., MAIN2, XLAST = 0; YLAST = 4

A label may be declared GLOBAL by the declaration:

e.g., GLOBAL LABEL BUFFER, POINT, LINE, SEND

Global labels are used for communication between routines, in particular to provide entry points. Thus a global label may be declared in several routines, but placed only in one routine. For example, BUFFER is declared global in Program 1 and Program 2, but is placed only in Program 2. The section on procedures explains how labels are used to define procedure entry points.

Address manipulations

SAL has very flexible facilities for handling addresses, as may be seen from the syntax. A label (global or local) may be used in any arithmetic or logical expression. When an expression is being evaluated the value taken for a label is the *address* assigned to that label, whereas the value taken for a variable is the *contents* of the address assigned to that variable; these are sometimes referred to as left and right hand values, respectively.[3]

Thus, if L is a label we could write:

A = L + 2

Indirect addresses are indicated with →. Thus:

→A = →D + 2

means take the value (see above) of A and use that as the address to store the value contained in the address given by the value of D, plus 2. The generality of the syntax (see appendix) permits the → operator to operate on expressions:

e.g., → (→A) = → (B + 2 * C)
→X = → (→ (→ (→ D))) + 1

To illustrate further the use of a label in an arithmetic expression and the → operator, suppose TABLE is the label marking the beginning of a table of addresses and an offset is in OFFSET, we could write:

GOTO → (TABLE + OFFSET)

since any arithmetic expression may follow a GOTO statment. The GOTO statement transfers control to the address given by the value of the expression following GOTO. A simple case is GOTO L, the value of the arithmetic expression L being the address of label L.

The address of a variable may be obtained with the PTR (pointer) operator:

e.g., A = PTR D

hence → (PTR A) is identical to A.

Data structures and components

The comprehensive address handling facilities described above plus the component feature provide the basic means for building and manipulating data structures. They are particularly suited to list, tree and ring type structures.

The component feature, similar to that in AED,[7] allows data to be read from or written to any field in any half-word in a contiguous block of store. The block of store is referenced by a pointer, which is any expression whose value is an address.

A complete half-word may be referenced using a component defined by a name and an offset:

e.g., ICOMP NEXT 0

This declares NEXT to be an integer component, with offset 0 as shown in Figure 1. Suppose P1 contains a pointer to a block of store, then a value may be set in the first half-word with a statement:

NEXT(P1) = 1024

Suppose we declare the next half-word (half-words are indicated by the increment O4, a rather unnatural Titan convention) to be a component SIZE:

e.g., ICOMP SIZE O4, O770 | O indicates octal.

This time we have further declared that the component

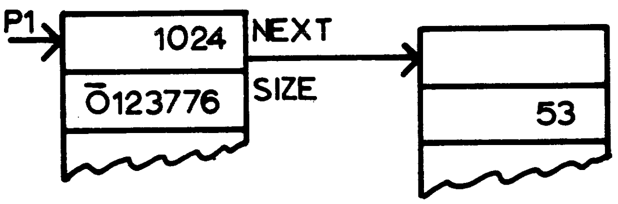


Figure 1—Components

SIZE refers only to that part of the word specified by the mask O770. When reading to or writing from such a field, only those bits corresponding to ones in the half-word are referenced.

If it is desired to shift data to the left before writing, or to the right after reading, then a count for the shift may also be specified:

e.g., ICOMP SIZE O4, O770, 3

If the half-word offset by O4 from P1 contained O123456, then the statement:

SIZE(P1) = O77

would change it to O123776.

The pointer may be specified by any arithmetic expression. We could, for example, set the SIZE component in the block pointed to by the NEXT component of P1, as shown in Figure 1, by the statement:

SIZE(NEXT(P1)) = 53

Reference to machine registers

As explained earlier the Titan computer has 128 index registers which are used as accumulators for integer arithmetic. These index registers may be referred to directly in any expression:

e.g., B27 = A + B26 + → B25 + NEXT(B24)

or if names have been declared for the index registers, then:

BLINE 24 POINTER, ADDR, ABC, RESULT

RESULT = A + ABC + ⇒ ADDR + NEXT (POINTER)

This feature allows efficient use of the machine registers while using high level statements, without the need for lapsing into assembly code.

Embedded assembly code and the lack of an Algol type block structure

Titan assembly code may be freely embedded at any position in a SAL program. Declared variables may be referenced, so providing communication between assembly code and other statements. Index registers may be referred to by their declared names. The four parts of a Titan assembly code instruction are:

Op. code Index register (Ba) Index register (Bm) Address or number

Suppose we want to send a block of 1000 words of data to the satellite computer attached to Titan

Data is in array J	
B1 = 1000	set B1 to number of words to be sent
B2 = 3000	set B2 to some address in satellite where data are to be sent
1013 1 0 J	send data to satellite using assembly code order
IF A = 2 THEN	carry on with rest of program

SAL does not have an Algol-like block structure. We could not convince ourselves that it would be any advantage to have one in this simple type of language. The advantage of being able to use the same name for different variables in different blocks is minimal. We had a strong incentive not to use a block structure with any automatic dynamic allocation of local variables within the blocks, as we wanted to be able to freely embed assembly code without restrictions. This includes the ability to jump using assembly code to any label or computed address. If there were a block structure the programmer would have to take special action when jumping into or out of a block, unless it was a purely lexicographic block structure as in PL360. An earlier section explained how the programmer may achieve the effect of automatic allocation by using external integer variables.

Procedures

As several conventions for passing arguments to procedures have grown up in the laboratory, we decided not to include any set way of defining or calling procedures in SAL. Any piece of code, however, may be considered as a procedure, with global or local labels marking the entry points (there is usually only one), specific code having to be written to pick up the arguments depending upon how they are passed. Calls to procedures must set up the arguments, save the return address, then jump to the entry point.

There is, however, an exception to this. SAL runs within the Mixed Language System (MLS) where assembly code, FORTRAN and SAL programs may be compiled separately but loaded and run together. Within MLS the standard method of passing arguments and calling procedures is the method required by FORTRAN. A general SAL feature allows calls to be made to procedures using this method, by preceding the call with MLS:

```
e.g., MLS SETUP(A, B, C)
      A = MLS CALCULATE(B)
```

the latter example having a value. The syntax allows the argument list to be preceded by any declared name. It is very useful permitting this name to be a variable, the value of which is the address of the entry point to the procedure. This is not quite as general as BCPL where the identity of the procedure may be given by the value of an expression.

```
e.g., INTEGER CHOICE
      CHOICE = → (TABLE + OFFSET)
                          | Table contains entry points
      MLS CHOICE(10)
```

This standard method of calling procedures could not be universally adopted for SAL since FORTRAN procedures and calling sequences are not pure procedures.

One of the strong points of SAL is its ability to communicate with programs written in other languages. To ease this problem further M. Richards has suggested declaring the names to be used for procedure calls to indicate to the compiler the type of call that must be set up:

```
e.g., FORTRAN SETUP, CALCULATE
      ALGOL CUBIC
      BCPL STRING, LIST
```

Calls to procedures need not then be preceded by a special word as at present (MLS).

```
e.g., SETUP(A,B,C)
      X = CUBIC(Y)
      LIST = LIST + 1 ; Z = LIST(P,Q,R)
```

Input/output

The input/output facilities are very Titan dependent so are not explained in any detail. Data transfers in Titan are arranged in "streams," enabling data to be read from any input device and sent to any output device in a generalized way. Facilities are provided to select an input or output stream, read or write a single alphanumeric character, a string of characters, or a binary character (12 bits). More complicated operations such as reading or writing numbers in different formats must be specifically programmed (probably by procedure call).

Implementation

The SAL compiler has been implemented using the Brooker-Morris compiler-compiler.[8] At the beginning of the project we chose to use the compiler-compiler as we believed that it would provide the fastest method of producing a SAL compiler and give us the flexibility to modify and experiment with the language. Hindsight tells us that we could almost certainly have written a compiler from scratch, in SAL, in the same time, especially allowing for the 2-3 month period learning to use the compiler-compiler. The SAL compiler produced by the compiler-compiler takes 16K and is not very fast (it generates about 30 instructions per second). This inefficiency is considerable since a SAL compiler written in SAL would probably be between 4 and 8K and operate a good deal faster. It is a serious criticism of the compiler-compiler that one cannot produce a more efficient compiler for a simple language like SAL. On the other hand, it does enable us to modify and experiment with the language very easily indeed.

No particular effort was made to make the generated code extremely efficient. Programs coded in assembly code are about 15 percent longer when written in SAL. Like an assembly code programmer, the SAL programmer has the ability to control the size and running efficiency of his program by careful attention to the code he writes. We refer here to the correct choice of language facility for a particular operation, and most definitely *not* to the sort of "tricks" which delight some assembly code programmers.

Future developments

The REPLACE feature

A simple macro feature built into SAL would enhance its usefulness. The proposed form of a macro definition is:

REPLACE *name of macro* BY *delimiter replacement text delimiter terminator.*

Arguments of the macro call would be indicated in the replacement text by some indication such as .A1 .A2 for arguments 1 and 2 and so on. For example, let us define a macro to enable us to add a word to a push down stack:

```
REPLACE STACK BY*
  →.A1 = .A2
   .A1 = .A1 + 1 ;*
```

A call STACK(S,X) would be then expanded to:

```
→S = X
 S = S + 1
```

This feature was not included as the compiler-compiler does not permit the input string to be modified before being processed by the syntax analyzer.

At present all macro generation must be done by a pre-pass using the ML/1 macro generator.[9]

An ENTRY feature

The most common way of saving a link when entering a procedure is to store it in index register B90. Hence having set up arguments, a procedure call would be:

```
e.g., B90 = RETN;GOTO PROC
      RETN, A = B
      -----
```

This practice means that a label must be defined for every return address, which is tedious, and reduces the readability of the code. A simple ENTER statement, which saved the return address in B90 automatically, then transferred control to the procedure would be most useful:

```
e.g., ENTER PROC
      A = B
      -----
```

CONCLUSIONS

SAL has been in daily use for about nine months. Programs written in SAL have included a command for the Titan multi-access system, routines to plot display files punched from our PDP7 on the Calcomp plotter attached to Titan, a general purpose ring structure package, and a system to define and transform three-dimensional pictures in Titan and transmit them to the PDP7 for display (Programs 1 and 2 are not part of this system). The language facilities, aided by a clear manual with a good index at the back[1] have proved most successful. We have achieved our objective in that system programs have proved easier to write, read and debug in SAL than in FORTRAN or assembly code. We are particularly pleased with SAL's ability to communicate with program segments written in other languages, and look forward to the inclusion of BCPL in the Titan Mixed Language System. The size of the compiler (16K) has proved a disadvantage, as the Titan multi-access system discriminates against large programs; on-line users would get faster response if the SAL compiler were smaller.

One man-year of effort has produced a system writing language which combines features normally associated with high level languages with the freedom and flexibility of assembly languages. Has the time not come when a SAL-like language should be the basic language provided for a computer rather than the conventional mnemonic assembly language? While being more complicated than an assembly language (to compile, *not* to code in) it is still simple, so the added time and core required for compilation should not be excessive. In any case, these are offset by the time likely to be saved in coding and debugging.

ACKNOWLEDGMENTS

I particularly want to acknowledge the splendid efforts of Heather Brown and Philip Cross with whom I have had many fruitful discussions about the language. Heather Brown implemented the first version of the compiler, always smilingly incorporating language changes as they arose. Philip Cross has done marvels to improve the efficiency of the compiler-compiler, and incorporated new language features found desirable in the light of experience.

APPENDIX

The syntax of SAL

Notation for describing the syntax

The notation is similar to that used by Brooker and

Morris.[8] An example of its use is the following description of an IF statement:

IF *expression condition expression* THEN *statements* END

As can be seen, a syntactic form is defined by concatenating its constituents. A constituent that is itself the name of a syntactic form is in italics. The remaining constituents are literals.

A special notation is needed to indicate parts of syntactic forms that may optionally be repeated and/or omitted. In this notation a constituent or series of constituents that may optionally be omitted is written:

[*constituents*?]

Constituents that may be repeated any desired number of times are written:

[*constituents**]

and constituents that may be omitted or repeated are written:

[*constituents**?]

The SAL IF statement has an optional ELSE clause, so is defined as:

IF *expression condition expression* THEN *statements* [ELSE *statements*?] END

Any number of names may be declared by an INTEGER declaration statement which is defined as:

INTEGER *declaration-name* [,*declaration-name**?] *terminator*

and an expression is defined as:

[*sign*?] *term* [*operator term**?]

A constituent is defined by writing all its possible forms in a list separated by commas. Particular forms are excluded by saying BUT NOT. Thus *operator* and *sign* as used above in the definition of an expression might be defined:

operator = +, −, *

sign = *operator*, BUT NOT *

Syntax definitions

A separator is defined as

separator = *space*, *tab*, *comment*, *continuation symbol*

To aid clarity, separator has been omitted from the syntax definitions below. It may be included anywhere in a SAL program.

Basic constituents

terminator = ;, *newline*
letter = A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,
Q,R,S,T,U,V,W,X,Y,Z
digit = 0,1,2,3,4,5,6,7,8,9
octal-digit = 0,1,2,3,4,5,6,7
start-letter = *letter*, .*letter*
alphanumeric = *letter*,*digit*,.
decimal-constant = [*digit**]
octal-constant = O[*octal-digit**] ,J [*octal-digit**]
b-line = B[*digit**] ,*name*
constant = *decimal-constant* [*octal-constant**?],
[*octal-constant**]
name = *start-letter* [*alphanumeric**?], BUT NOT
B *digit*, O *digit*, J *digit*, END,ELSE
label = *name*
array-variable = *name*(*expression*)
declaration-name = *name*, *name*(*constant*)
component = *name*(*expression*)
pointer = PTR *variable*, PTR *array-variable*
indirect-address = → *b-line*, → *variable*, → *array-variable*, → (*expression*)
character-string = : :[*any-character**?] [*newline*?]: :
sign = +, −
operator = +,−,*,/
term = *constant*, *b-line*, *label*, *variable*, *array-variable*, *component*, *pointer*, *indirect-address*, (*expression*)
expression = [*sign*?] *term* [*operator term**?]
address-term = *constant*, *label*, *variable*, *array-variable*,*,.
address-expression = [*sign*?] *address-term* [*sign address-term**?]
destination = *b-line*, *label*, *variable*, *array-variable*, *component*, *indirect-address*
shift = SHIFTLA, SHIFTLC, SHIFTRA,
SHIFTRC
condition = EQ, =, NE, GT, >, LT, <, LE

Declarations

GLOBAL LABEL *name* [,*name**?] *terminator*
INTEGER *declaration-name* [,*declaration-name**?] *terminator*

GLOBAL INTEGER *declaration-name* [,*declaration-name**?] *terminator*
RELSTART INTEGER *b-line constant terminator*
INTEGER *b-line declaration-name* [,*declaration-name**?] *terminator*
REAL *declaration-name* [,*declaration-name**?] *terminator*
B LINE [*b-list**] *terminator*
b-list = [*bnum*?] *name* [,*name**?]
bnum = *b-line*, [*digit**]
COMMON [*c-list**] *terminator*
c-list = / [*name*?] / *declaration-name* [,*declaration-name**?]
ICOMP *name constant* [,*constant*?] [,*constant*?] *terminator*

Preset (compile-time) declaration

SET *declaration-name* [TO *declaration-name*?] = *address-expression* [,*address- expression**?] *terminator*

Compile time constants

name IS *address-expression terminator*

Placing a label

label,

Assignment statements

destination = *expression terminator*
destination = *character-string terminator*
destination = *term* AND *term terminator*
destination = *term* OR *term terminator*
destination = NOT *term terminator*
destination = *term shift term terminator*

Transfer of control, iterative and conditional statements

GOTO *expression terminator*
STOP *terminator*
FOR *destination* = *expression* STEP *expression* UNTIL *expression* DO *statements terminator* REPEAT *terminator*
WHILE *expression condition expression* DO *statements terminator* REPEAT *terminator*
IF *expression condition expression* THEN *statements* [ELSE *statements*?] END

Machine code statements

op ba bm address-expression terminator

Setting up half-words of data

HWDS *address-expression* [,*address-expression**?] *terminator*

Reserving space

RESERVE *address-expression terminator*

Input and output statements

Omitted as they are so Titan dependent. They provide for setting up and manipulating input and output streams and reading or writing a binary number, a character, or a "record" (a string of characters terminated by a "carriage control character". For example, a "newline").

Procedure calls using the MLS conventions

MLS *name* (*argument* [,*argument**?] *terminator*
destination = MLS *name* (*argument* [,*argument**?]) *terminator*

End of a routine

FINISH *terminator*

REFERENCES

1 H BROWN
SAL user's manual
University Mathematical Laboratory Cambridge June 1968

2 A EVANS
PAL—A language designed for teaching programming linguistics
Proc 23rd National ACM Conference 1968

3 M RICHARDS
BCPL: A tool for compiler writing and systems programming
Proc S J C C 1969 (this issue)

4 N WIRTH
PL360 A programming language for the 360 computers
J ACM Vol 15 No 1 January 1968

5 R HAY J F RULIFSON
MOL940: Preliminary specification for an Algol-like machine oriented language for the SDS 940
Interim Tech Report 2 Project 5890 Stanford Research Institute March 1968

6 P NAUR Editor
Revised report on the algorithmic language Algol 60
Computer Journal Vol 5 p 349 January 1963

7 D T ROSS
AED-0 programming manual
Preliminary Release
Massachusetts Institute of Technology October 1964

8 R A BOOKER et al
The compiler compiler
Annual Review in Automatic Programming
Pergamon Press 1963

9 P J BROWN
The ML/1 macro processor
C ACM Vol 10 No 10 October 1967

BCPL: A tool for compiler writing and system programming

by MARTIN RICHARDS*

University Mathematical Laboratory
Cambridge, England

INTRODUCTION

The language BCPL[1] (Basic CPL) was originally developed as a compiler writing tool and as its name suggests it is closely related to CPL[2,3] (Combined Programming Language) which was jointly developed at Cambridge and London Universities. BCPL adopted much of the syntactic richness of CPL and strived for the same high standard of linguistic elegance; however, in order to achieve the efficiency necessary for system programming its scale and complexity is far less than that of CPL. The most significant simplification is that BCPL has only one data type—the binary bit pattern—and this feature alone gives BCPL a characteristic flavour which is very different of that of CPL and most other current programming languages.

BCPL has proved itself to be a very useful compiler writing tool and it also has many qualities which make it highly suitable for other system programming applications.

We will first outline the general structure of BCPL and later discuss how well it is suited to applications in the fields of compiler writing and system programming.

The language

BCPL has a simple underlying semantic structure which is built around an idealised object machine. This method of design was chosen in order to make BCPL easy to define accurately and to facilitate machine independence which is one of the fundamental aims of the language.

* The work was started while the author was employed by Massachusetts Institute of Technology. It was supported, in part, by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).

The most important feature of the object machine is its store and this is represented diagrammatically in Figure 1. It consists of a set of numbered boxes (or storage cells) arranged so that the numbers labelling adjacent cells differ by one. As will be seen later, this property is important.

Each storage cell holds a binary bit pattern called an Rvalue (or Right hand value). All storage cells are of the same size and the length of Rvalues is a constant of the implementation which is usually between 24 and 36 bits. An Rvalue is the only kind of object which can be manipulated directly in BCPL and the value of every variable and expression in the language will always be an Rvalue.

Rvalues are used by the programmer to model abstract objects of many different kinds such as truth values, strings and functions, and there are a large number of basic operations on Rvalues which have been provided in order to help the programmer model the transformation of his abstract objects. In particular, there are the usual arithmetic operations which operate on Rvalues in such a way that they closely model integers. One can either think of these operations as ones which interpret their operands as integers, perform the integer arithmetic and convert the result back into the Rvalue form, alternatively one may think of them as operations which work directly on bit patterns and just happen to be useful for representing integers. This latter approach is closer to the BCPL

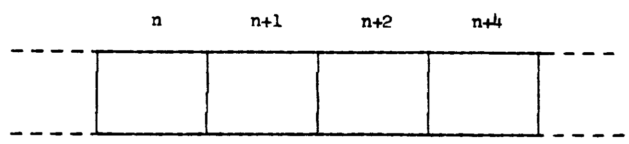


Figure 1—The machine's store

philosophy. Although the BCPL programmer has direct access to the bits of an Rvalue, the details of the binary representation used to represent integers are not defined and he would lose machine independence if he performed non numerical operations on Rvalues he knows to represent integers.

An operation of fundamental importance in the object machine is that of Indirection. This operation has one operand which is interpreted as an integer and it locates the storage cell which is labelled by this integer. This operation is assumed to be efficient and, as will be seen later, the programmer may invoke it from within BCPL using the *rv* operator.

Variables and manifest constants

A variable in BCPL is defined to be a name which has been associated with a storage cell. It has a value which is the Rvalue contained in the cell and it is called a variable since this Rvalue may be changed by an assignment command during execution. Almost all forms of definition in BCPL introduce variables. The only exception is the *manifest* declaration which is used to introduce manifest constants.

A manifest constant is the direct association of a name with an Rvalue; this association takes place at compile time and remains constant throughout execution. There are many situations where manifest constants can be used to improve readability with no loss of runtime efficiency.

Lvalues and modes of evaluation

As previously stated each storage cell is labelled by an integer; this integer is called the Lvalue (or Left hand value) of the cell. Since a variable is associated with a storage cell, it must also be associated with an Lvalue and one can usefully represent a variable diagrammatically as in Figure 2.

Within the machine an Lvalue is represented by a binary bit pattern of the same size as an Rvalue and so an Rvalue can represent an Lvalue directly. The process of finding the Lvalue or Rvalue of a variable is called Lmode or Rmode evaluation respectively. The idea of mode of evaluation is useful since it applies to expressions in general and can be used to clarify the semantics of the assignment command and other features in the language.

Simple assignment

The syntactic form of a simple assignment command is:

E1 := E2

where E1 and E2 are expressions. Loosely, the meaning of the assignment is to evaluate E2 and store its value in the storage cell referred to by E1. It is clear that the expressions E1 and E2 are evaluated in different ways and hence there is the classification into the two modes of evaluation. The left hand expression E1 is evaluated in Lmode to yield the Lvalue of some storage cell and the right hand side E2 is evaluated in Rmode to yield an Rvalue; the contents of the storage cell is then replaced by the Rvalue. This process is shown diagrammatically in Figure 3. The only expressions which may meaningfully appear on the left hand side of an assignment are those which are associated with storage cells, and they are called Ltype expressions.

The terms Lvalue and Rvalue derive from consideration of the assignment command and were first used by Strachey in the CPL reference manual.

The lv operator

As previously stated an Lvalue is represented by a binary bit pattern which is the same size as an Rvalue. The *lv* expression provides the facility of accessing the

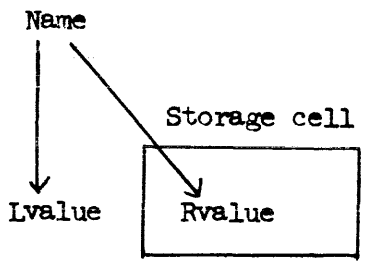


Figure 2—The form of a variable

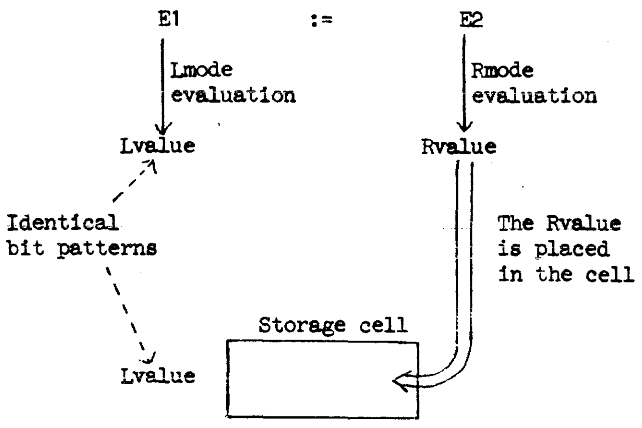


Figure 3—The process of assignment

Lvalue of a storage cell and, as will be seen, this ability is very useful.

The syntactic form of an *lv* expression is:

lv E

where E is an Ltype expression. The evaluation process is shown in Figure 4. The operand is evaluated in Lmode to yield an Lvalue and the result is a bit pattern identical to this Lvalue. The *lv* operator is exceptional in that it is the only expression operator to invoke Lmode evaluation, and indeed in all other contexts, except the left hand side of the assignment, expressions are evaluated in Rmode.

The rv operator

The *rv* operator is important in BCPL since it provides the underlying mechanism for manipulating vectors and data structures; its operation is one of taking the contents (or Rvalue) of a storage cell whose address (or Lvalue) is given.

The syntactic form of an *rv* expression is as follows:

rv E

and its process of evaluation is shown diagrammatically in Figure 5. The operand is evaluated in Rmode and then the storage cell whose Lvalue is the identical bit pattern is found. If the *rv* expression is being evaluated in Rmode, then the contents of the cell is the result; however, it is also meaningful to evaluate it in Lmode, in which case the Lvalue of the cell is the result. An *rv* expression is thus an Ltype expression and so may

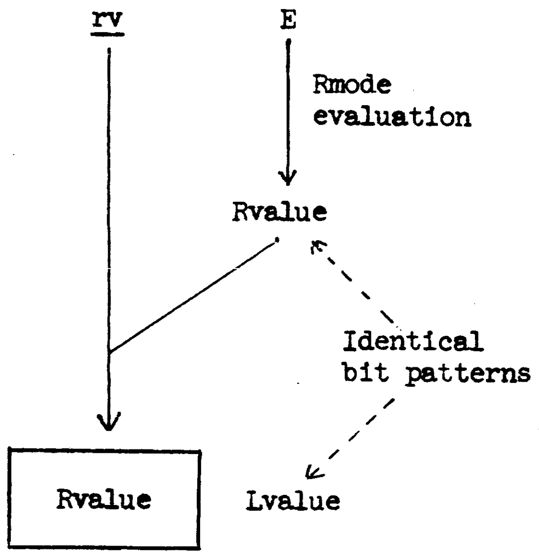


Figure 5—The evaluation of an RV expression

appear on the left hand side of an assignment command, as in:

rv p := t

and one can deduce that this command will update the storage cell pointed to by p with the Rvalue of t.

Data structures

The considerable power and usefulness of the *rv* operator can be seen by considering Figure 6. This

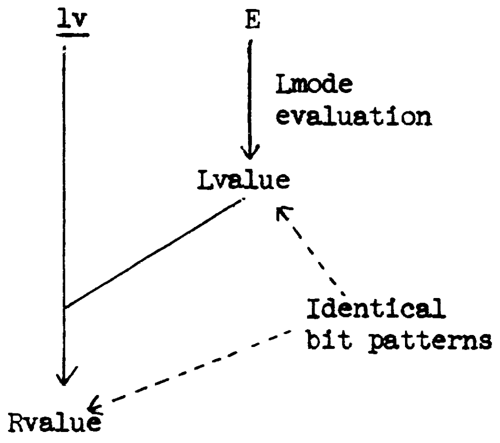


Figure 4—The evaluation of an LV expression

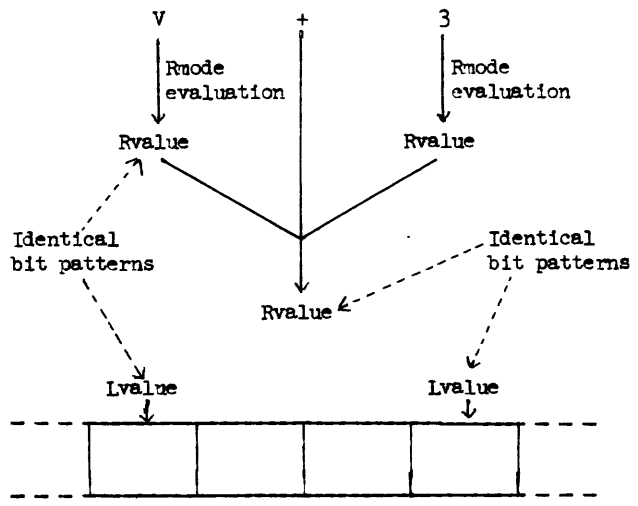


Figure 6—An intrepretation of V + 3

diagram shows a possible interpretation of the expression V + 3. Some adjacent storage cells are shown and the left most one has an Lvalue which is the same bit pattern as the Rvalue of V. One will recall that an Lvalue is really an integer and that Lvalues of adjacent cells differ by one, and thus the Rvalue of V + 3 is the same bit pattern as the Lvalue of the rightmost box shown in the diagram. If the operator *rv* is applied to V + 3, then the contents of that cell will be accessed. Thus the expression:

$$rv\ (V + i)$$

acts very like a vector application, since, as i varies from zero to three, the expression refers to the different elements of the set of four cells pointed to by V. V can be thought of as the vector and i as the integer subscript.

Since this facility is so useful, the following syntactic sugaring is provided:

E1 ↓ E2 is equivalent to *rv* (E1 + E2)

and a simple example of its use is the following command:

V ↓ (i + 1) : = V ↓ i + 2

One can see how the *rv* operation can be used in data structures by considering the following:

V ↓ 3 ≡ *rv* (V + 3) by definition
≡ *rv* (3 + V) since + is commutative
≡ 3 ↓ V

Thus V ↓ 3 and 3 ↓ V ares emantically equivalent; however, it is useful to attach different interpretations to them. We have already seen an interpretation of V ↓ 3 so let us consider the other expression. If we rewrite 3 ↓ V as Xpart ↓ V where Xpart has value 3, we can now conveniently think of this expression as a selector (Xpart) applied to a structure (V). This interpretation is shown in Figure 7.

By letting the elements of structures themselves be structures it is possible to construct compound data structures of arbitrary complexity. Figure 8 shows a structure composed of integers and pointers.

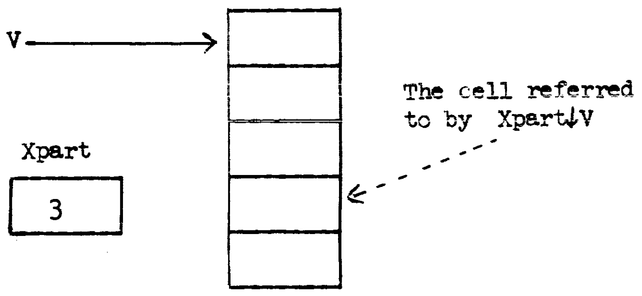


Figure 7—An interpretation of X part ↓ V

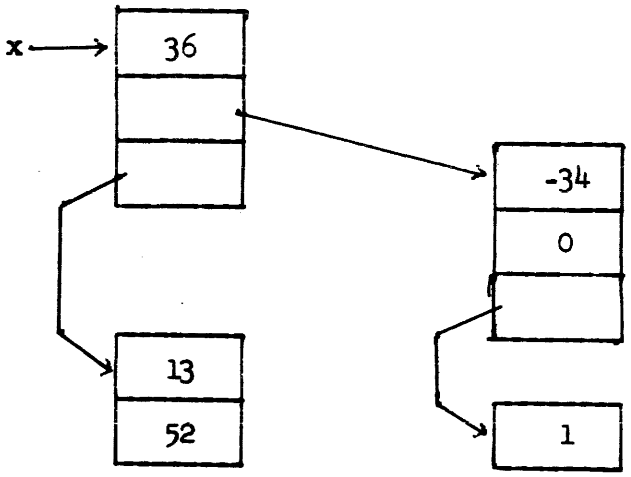


Figure 8—A structure of integers and pointers

Data types

The unusual way in which BCPL treats data types is fundamental to its design and thus some discussion of types is in order here. It is useful to introduce two classes:

a. Conceptual types
b. Internal types

The Conceptual type of an expression is the kind of abstract object the programmer had in mind when he wrote the expression. It might be, for instance, a time in milliseconds, a weight in grams, a function to transform feet per second to miles per hour, or it might be a data structure representing a parse tree. It is, of course, impossible to enumerate all the possible conceptual types and it is equally impossible to provide for all of them individually within a programming language. The usual practice when designing a language is to select from the conceptual types a few basic ones and provide a suitable internal representation together with enough basic operations. The term Internal type refers to any one of these basic types and the intention is that all the conceptual types can be modelled effectively using the internal types. A few of the internal types provided in a typical language, such as CPL, are listed below:

real

integer

label

integer function

(*real*, *Boolean*) *vector*

Much of the flavour of BCPL is the result of the conscious design decision to provide only one internal type, namely: the binary bit pattern (or Rvalue). In order to allow the programmer to model any conceptual type many useful primitive operations have been provided. For instance, the ordinary arithmetic operators +, −, * and / have been defined for Rvalues in such a way as to model the integer operations directly. The six standard relational operators have been defined and a complete set of bit manipulating operations provided. In addition, there are some stranger bit pattern operations which provide ways of representing functions, labels and, as we have already seen, vectors and structures. All these operations are uniformly efficient and can usually be translated into one or two machine instructions.

The most important effects of designing a language in this way can be summarized as follows:

1. There is no need for type declarations in the language, since the type of every variable is already known. This helps to make programs concise and also simplifies such linguistic problems as the handling of actual parameters and separate compilation.
2. It gives the language nearly the same power as one with dynamically varying types (e.g., PAL[4]) and yet retains the efficiency of a language (like FORTRAN[5]) with manifest types; for, although the internal type of an expression is always known by the compiler, its conceptual type can never be. For instance it may depend on the values of variables within the expression, as in the vector application $V \downarrow i$, since the elements of a vector are not necessarily of the same conceptual type. One should note that in languages (such as ALGOL[6] and CPL) where the elements of vectors must all have the same type, one needs some other linguistic device in order to handle dynamically varying data structures.
3. Since there is only one internal type there can be no automatic type checking and it is possible to write nonsensical programs which the compiler will translate without complaint. This slight disadvantage is easily outweighed by the simplicity, power and efficiency that this treatment of types makes possible.

Syntactic features of BCPL

One of the design criteria of BCPL was that it should be a useful system programming tool and it was felt that high readability was of extreme importance. The readability of a program largely depends on the skill and style of the programmer; however, his task is greatly simplified if he is using a language with a rich set of expressive but concise constructions and if all the little syntactic details have been well thought out.

The syntax of BCPL is based on the syntax of CPL and, although the underlying semantics of the two languages are very different, they look superficially alike.

One of the most important requirements necessary before one can obtain a reasonable degree of readability is an adequate character set which contains both capital and small letters. A comparison has been made between two hardware versions of the same large BCPL program, one using nearly the full ASCII character set and the other using the same set without any small letters. Although there is no accurate measure of readability, it was agreed by all who made the comparison that the difference between the two versions was very significant. The lengthening of identifiers to avoid clash of names, and the fact that system words and identifiers were no longer distinct both increased the difficulty of reading the program. There are satisfactory implementations of BCPL using a restricted character set, but such a set should only be used where absolutely necessary.

BCPL follows CPL in the selection of commands. There are the three basic commands: assignments, routine commands and jumps, and there is the large variety of syntactic constructions to control the flow of control within an algorithm; some example forms are given below:

test E *then* C *or* C

if E *do* C

unless E *do* C

until E *do* C

while E *do* C

C *repeatuntil* E

C *repeatwhile* E

C *repeat*

for Name = E *to* E *do* C

where E denotes any expression and C any command. A very useful pair of additional commands are

a. *break* which causes a jump out of the smallest enclosing loop command, and
b. *return* which causes a return from the current routine.

One of the most noticeable ways in which this large selection of constructions improves readability is by the considerable reduction in the need for labels and goto commands. For instance, the BCPL compiler itself consists of 88 pages of BCPL program and contains only 29 labels which is about one label per three pages of program. It is interesting to see how experienced FORTRAN programmers fill their first few BCPL programs with labels and how their programming style improves as they gain facility.

The BCPL syntax for declarations and definitions is far simpler than the corresponding syntax in CPL; this is mainly due to the elimination of types from the language, and the lower emphasis placed on declarations in BCPL.

The purpose of a declaration in BCPL is threefold:

a. To introduce a name and specify its scope.
b. To specify its extent.
c. To specify its initial value.

The *scope* of a name is the textual region of program in which it may be used to refer to the same data item; this region is usually a block or the body of a function or routine, and it depends on the way in which the name was declared. The *extent* of a variable is the time through which it exists and is associated with a storage cell. Throughout the extent of a variable, its Lvalue remains constant and its Rvalue is only changed by assignment. Most forms of declaration initialize the variables they define, as in:

let f (t) = 2*t + 3

let *x* = 36 + f(4)

In this example, the variable f is initialized to a value which represents the function defined, and x is initialized to 47.

In BCPL, variables may be divided into two classes:

1. *Static variables*
The extent of a static variable is the entire execution time of the program; the storage cell is allocated prior to execution and continues to exist until execution is complete.
2. *Dynamic variables*
A dynamic variable is one whose extent starts when its declaration is executed and continues until execution leaves the scope of the variable. Dynamic variables are particularly useful when using recursive functions and routines. The kind of variable declared depends on the form of declaration used; out of the nine methods of declaring names in BCPL, five declare static variables, three produce dynamic variables and the remaining one declares manifest constants.

Function and routine calls

In BCPL as in CPL, there is a rigorous distinction between expressions and commands which shows itself in the syntax of the language; it also causes the semantic separation of functions from routines. In many respects functions and routines are rather similar; however, a function application is a kind of expression and yields a result, whereas a routine call is a kind of command and does not.

The syntactic form of both function applications and routine calls is as follows:

E1(E2, E3 . . . ., En)

where E1 to En all denote expressions. The expressions E2 to En are called actual parameters. The evaluation process is as follows:

1. All the expressions E1 to En are evaluated in Rmode to yield Rvalues.
2. A set of n-1 adjacent new storage cells are found and the values of E2 and En are stored in them.
3. The function or routine corresponding to the value of E1 is found and the formal parameters are associated with the cells containing the arguments. This association is performed from left to right and there is no need for the number of actual parameters to equal the number of formals.
4. The body of the function or routine is then executed in the new environment.
5. When the body has been completely evaluated, execution returns to the call. For a routine call there is no result and execution is now complete; however, for a function application there is a result which is the Rvalue of the function body.

All functions and routines in BCPL are automatically recursive and so, for instance, one can call a function while an activation of that function is already in existence. In order to allow for recursion and yet maintain very high execution efficiency, the restriction has been imposed that all free variables of both functions and routines must be static. Randell and Russell[7] give a good description of the kind of mechanism normally required for recursive calls in ALGOL; however, with this restriction, a recursive call in BCPL can be very efficient.

The mobility of the BCPL Compiler

A program is machine independent if it can be transferred from one machine to another without change. Complete machine independence is rarely achieved; however, it is a goal well worth striving for. For large systems, mobility is often a more useful measure than machine independence. Mobility is a measure of how easy it is to transfer a system from one machine to another; it differs from machine independence because the program often requires some redesign and reprogramming. For example, when moving a compiler from one machine to another it is necessary to rewrite the code generator and usually part of the lexical analyzer. Writing a compiler in a machine independent language is an important factor in obtaining mobility but it does not ensure it; it is at least as important to design the overall structure of the compiler with mobility in mind. The BCPL compiler has been designed with this aim and has been transferred successfully to seven other machines without much difficulty.

BCPL is a simple language to compile and it has a straightforward compiler written in BCPL. The compiler is easy to follow and it produces fairly good object code at an acceptably fast speed. Its general structure is shown in Figure 9. The rectangular boxes represent the different logical parts of the compiler and the round boxes the various intermediate forms the BCPL program takes while it is being compiled. These intermediate forms will be briefly sketched by considering the transformations of the program shown in Figure 10.

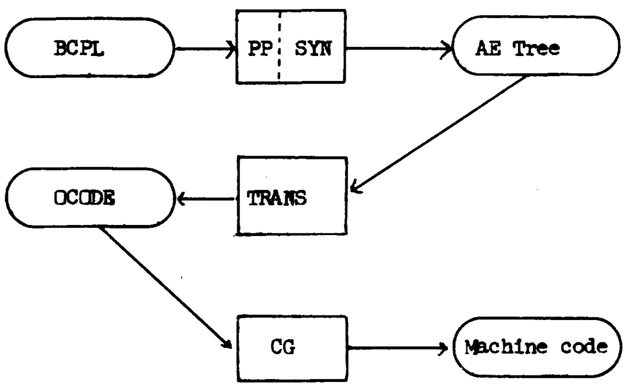


Figure 9—The structure of the BCPL compiler

The input form of the program is first transformed into an internal tree structure called the Applicative Expression Tree (AE Tree); this is done by the syntax analyzer which is composed of a set of machine independent parsing functions (SYN) and a lexical analyzer routine (PP). The AE tree structure for our example program is shown in Figure 11.

The AE tree is then translated by Trans into an intermediate object code called OCODE. OCODE was specially designed for BCPL and it is a simple language whose statements cause basic transformations on an imaginary stack machine; it was designed to be as machine independent as is practical to keep the changes to Trans to a minimum when the compiler is moved to a new machine.

The Code generator translates OCODE statements into the machine code of the object machine. The implementor is free to choose between relocatable binary and assembly language, and it is usually found that the ability to generate both is very valuable.

```
$(  let x = 0
    if x ne 2 do  x := x + 2
    finish   $)
```

Figure 10—An example BCPL program

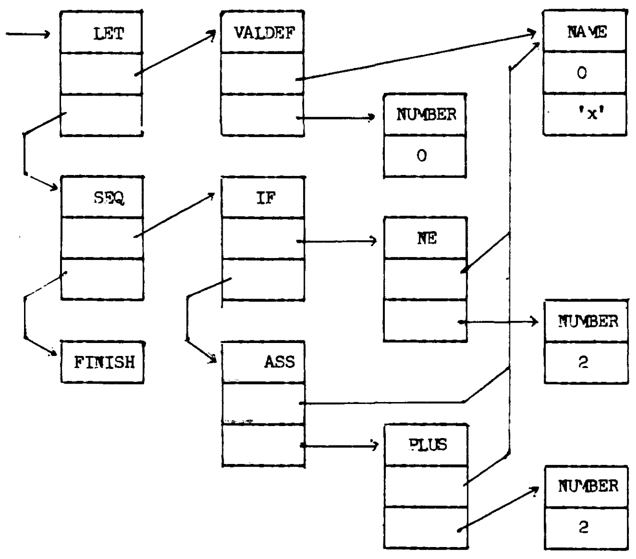


Figure 11—The AE tree form of Figure 10

In order to transfer BCPL to a new machine, one must choose a suitable strategy and this usually depends on the locality of the machines, their basic compatibility and the facilities available on the recipient machine. The basic process is to write both a code generator for the new machine and a suitable machine code interface with the new operating system; one then modifies and corrects the few machine dependencies in the syntax analyser and Trans, and finally compiles the new compiler using the new code generator. The process is complicated by the fact that the work cannot be carried out entirely on one machine. In practice, the more work that can be done on the donor machine the better; however, one often has no direct access to that machine and a different strategy must be applied. In this situation it is usually better to implement a temporary code generator for the recipient machine in some standard language such as FORTRAN or SNOBOL[8] and then use it to compile the OCODE files of the syntax analyzer and translation phase of the compiler. One can then construct a temporary BCPL compiler on the new machine and use it to compile itself. The compiler can then be polished to fit well in its new operating environment.

The cost of transferring BCPL depends largely on the computing facilities available, and one can expect it to be between one and five man months.

The use of BCPL for compiler writing

There are many reasons why BCPL is suitable for compiler writing and probably one of the most important of these is the ease of programming in the language. This together with its inherently high readability combine to make BCPL a very flexible language. The richness and variety of useful commands are valuable and the built in recursion is almost essential. In order to see how well these features may be used together we will consider a short excerpt from the BCPL compiler. Figure 12 shows the overall structure of the main part of translation phase. The directive *get* 'HEAD2' causes the compiler to insert a file of BCPL text and compile it with what follows. This insertion facility is very useful when co-ordinating many separate parts of a large program. This example shows how the *switchon* command may be used with manifest constants to good effect. It is executed by evaluating the controlling expression H1 ↓ x and then jumping to the case corresponding to the value found. The expression appearing in the case labels are manifest constants and denote the possible AE tree operators that Trans must deal with (the constants LET, TEST and REPEAT are declared in the inserted file HEAD2. If the value of H1 ↓ x does not correspond to any case then execution continues at the default label. Since all the case constants are known by the compiler it is possible to implement the switch very efficiently, even constructing a hash table if this method of switching is appropriate. This combination of manifest constants and *switchon* commands is very effective and it has been used frequently in the BCPL compiler.

```
get 'HEAD2'

let Trans(x) be

    $(1 switchon H1↓x into

        $( default: return

            case LET:      - - -

                           - - -

            case TEST:     - - -

                           - - -

            case REPEAT:   - - -

                           - - -      $)1
```

Figure 12—The structure of Trans

Recursion is also very useful in many compiling situations particularly in parts concerned with the creation and analysis of tree structures. Figure 13 is a detailed excerpt, again taken from Trans, and it provides an example of a typical use of recursion. The section of program shown is used to translate the AE tree form of a *test* command into OCODE. The variable x is the formal parameter of Trans and it points to a

```
case TEST: $( let L, M = Nextparam(), Nextparam()

              Jumpcond(H2↓x, false, L)

              Trans(H3↓x)

              Compjump(M)

              Complab(L)

              Trans(H4↓x)

              Complab(M)

              return     $
```

Figure 13—A detail from the body of Trans

TEST node. The form of this node is shown in Figure 14; the pointers to E, C1 and C2 represent the branches to nodes for the Boolean expression and two alternative commands of the *test* command. These components can be accessed by the expressions H2 ↓ x, H3 ↓ x, and H4 ↓ x, respectively. To execute a test command, first the Boolean expression is evaluated and then, if the result is true, the first alternative is executed, alternatively the second. The general form of the object code is as follows:

1. Code to evaluate E.
2. Code to jump to label L if the result is false.
3. Code corresponding to the translation of C1.
4. An unconditional jump around the code for C2.
5. A directive to set label L.
6. Code for C2.
7. A directive to set the label used in step 4.

As can be seen the program to generate this code is very straightforward. First, two local variables L and M are declared for the two labels. The call for Jumpcond then compiles the code for steps 1 and 2. Its first argument is the Boolean expression of the *test* command and the other arguments specify the kind of conditional jump required and the label to jump to. The next statement is a call for Trans which will compile the first alternative C1. This is an example of the recursive use of Trans. The calls for Compjump and Complab generate code corresponding to steps 4 and 5, and then there is a second recursive call for Trans to translate C2. Finally, a directive to set label M is compiled and then, since the *test* command has now been completely translated, a return is made to the current call of Trans.

One should note how convenient it is not to have to declare the types of the variables such as x, L and M, and one should also note how well the use of manifest constants, switchon commands, recursion and simple data structures combine to produce a very effective and readable means of writing this part of the compiler. Although there is considerable variance in the style of programming used in the different parts of the compiler, the facilities and syntactic qualities of BCPL have made it possible to achieve this high standard of simplicity and readability throughout.

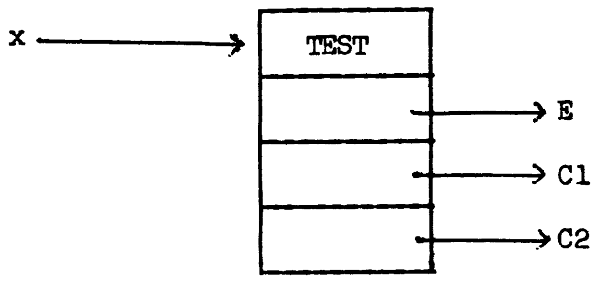


Figure 14—The AE tree form of a test command

The way in which BCPL treats data types allows the programmer great freedom to organize his symbol tables, property lists, tree structures and stacks in the most suitable fashion for his particular application. Admittedly BCPL only provides the basic operations and the compiler writer must write his own system, but this is easy to do and he does not suffer the disadvantage of having to use a system in which inappropriate design decisions have already been made. The philosophy of BCPL is not one of a tyrant who thinks he knows best and lays down the law on what is and what is not allowed; rather, BCPL acts more as a servant offering all his services to the best of his ability without complaint even when confronted with apparent nonsense. The programmer is always assumed to know what he is doing and he is not hemmed in by petty restrictions. Machine code programmers tend to like the way in which BCPL combines the advantages of a high level language with the power to do address arithmetic and to be able to manipulate binary bit patterns without invoking a great weight of expensive machinery.

When planning and writing a compiler in a commercial environment one must make a compromise between the quality of the product and its cost. The quality of a compiler is affected by many factors such as its size, its compile speed, the efficiency of the object code produced, the usefulness of the error diagnostics, the accuracy and quality of its documentation, its maintainability and in some cases its flexibility and mobility. Only the first two of these are directly improved by writing the compiler in a more efficient language, while the others tend to suffer because the compiler is harder to write. Although efficiency is important in a compiler writing language, this consideration should not totally dominate its design. The author believes that the compromise in the design of BCPL between efficiency and linguistic effectiveness is near optimal for compilers of medium to large scale languages especially if flexibility is required.

REFERENCES

1 M RICHARDS
The BCPL reference manual
Memorandum-69/1 The University Mathematical Laboratory Cambridge England January 1969

2 J BUXTON J C GRAY D PARK
CPL elementary programming manual
Edition II The University Mathematical Laboratory Cambridge England 1966

3 C STRACHEY (editor)
CPL working papers
Cambridge University Mathematical Laboratory and London Institute of Computer Science 1965

4 A EVANS JR
A language for teaching programming linguistics
Proc 23rd National ACM Conference 1968

5 IBM Reference Manual
709/7094 FORTRAN Programming System
Form C28-6054-2

6 P NAUR (editor)
Revised report on the algorithmic language ALGOL 60
The Computer Journal Vol 5 January 1963 349

7 B RANDELL L J RUSSELL
ALGOL 60 implementation
Academic Press 1964

8 D J FARBER et al
SNOBOL, a string-manipulation language
J ACM Vol 11 No 1 January 1964

APPENDIX

The syntax given below is Bachus Naur Form with the following extensions:

1. For improved readability, the syntactic categories for expressions, commands and definitions (namely E, C and D) are not surrounded by meta linguistic brackets.
2. The symbols { and } are used to indicate repetition, for example:

   E { , E }$_0^\infty$ means

   E | E, E | E, E, E | . . . etc

This syntax is ambiguous and is simply intended to list all the syntactic contructions available.

The canonical syntax of BCPL

E ::= ⟨name⟩ | ⟨stringconst⟩ | ⟨charconst⟩ |
⟨number⟩ | *true* | *false* | (E) | *valof* ⟨block⟩ |
lv E | *rv* E | E(⟨E list⟩) | E() | E ⟨diadic op⟩ E|
⟨monadic op⟩ E | E → E, E |
table ⟨constant⟩ {, ⟨constant⟩ }$_0^\infty$

⟨diadic op⟩ ::= ↓ | * | / | *rem* | + | − |
= | ≠ | *ls* | *gr* | *le* | *ge* |
lshift | *rshift* | ∧ | ∨ | ≡ | ≢

⟨monadic op⟩ ::= + | − | *not*

⟨E list⟩ ::= E {, E}$_0^\infty$

⟨constant⟩ ::= E

C ::= ⟨E list⟩ : = ⟨E list⟩ | E(⟨E list⟩) | E() |
goto E | ⟨name⟩ : C | *if* E *do* C | *unless* E *do* C |
while E *do* C | *until* E *do* C | C *repeat* |
C *repeatuntil* E | C *repeatwhile* E |
test E *then* C *or* C | *break* | *return* | *finish* |
resultis E | *for* ⟨name⟩ = E *to* E *do* C |
switchon E *onto* ⟨block⟩ | *case* ⟨constant⟩ : C |
default : C | ⟨block⟩ | ⟨empty⟩

D ::= ⟨name⟩ (⟨FPL⟩) = E | ⟨name⟩ (⟨FPL⟩) *be* C |
⟨name list⟩ = ⟨E list⟩
⟨name⟩ = *vec* ⟨constant⟩

⟨FPL⟩ ::= ⟨name list⟩ | ⟨empty⟩

⟨name list⟩ ::= ⟨name⟩ {, ⟨name⟩ }$_0^\infty$

⟨block⟩ ::= $(⟨block body⟩ $)

⟨block body⟩ ::= C{; C}$_0^\infty$|
⟨declaration⟩ }' $_0^\infty${; C}$_0^\infty$

⟨declaration⟩ ::= *let* D {*and* D}$_0^\infty$| *static* ⟨decl body⟩ |
manifest ⟨decl body⟩ |
global ⟨decl body⟩

⟨decl body⟩ ::= $(⟨C def⟩ {; ⟨C def⟩}$_0^\infty$$)

⟨C def⟩ ::= ⟨name⟩ : ⟨constant⟩ |
⟨name⟩ = ⟨constant⟩

⟨program⟩ ::= ⟨block body⟩

EXDAMS—EXtendable Debugging and Monitoring System*

by R. M. BALZER

The RAND Corporation
Santa Monica, California

INTRODUCTION

With the advent of the higher-level algebraic languages, the computer industry expected to be relieved of the detailed programming required at the assembly-language level. This expectation has largely been realized. Many systems are now being built in higher-level languages (most notably MULTICS).[1]

However, our ability to debug programs has not advanced much with our increased use of the higher-level languages. As Evans and Darley[2] point out:

> We find that, broadly speaking, a close analog of almost every principal assembly-language debugging technique exists in at least one debugging system pertaining to some higher-level language. However, on-line debugging facilities for higher-level languages are in general less well-developed and less widely used (relative to the use of the languages) than their assembly-language counterparts.

We have, in general, merely copied the on-line assembly-language debugging aids, rather than design totally new facilities for higher-level languages. We have neither created new graphical formats in which to present the debugging information, nor provided a reasonable means by which users can specify the processing required on any available debugging data.

These features have been largely ignored because of the difficulty of their implementation. The debugging systems for higher-level languages are much more complex than those for assembly code. They must locate the symbol table, find the beginning and end of source-level statements, and determine some way to extract the dynamic information—needed for debugging—about the program's behavior, which is now hidden in a sequence of machine instructions rather than being the obvious result of one machine instruction. Is it any wonder that, after all this effort merely to create a minimal environment in which to perform on-line higher-level language debugging, little energy remained for creating new debugging aids that would probably require an increased dynamic information-gathering capability.

EXDAMS (EXtendable Debugging And Monitoring System) is an attempt to break this impasse by providing a single environment in which the users can easily add new on-line debugging aids to the system one-at-a-time without further modifying the source-level compilers, EXDAMS, or their programs to be debugged. It is hoped that EXDAMS will encourage the creation of new methods of debugging by reducing the cost of an attempt sufficiently to make experimentation practical. At the same time, it is similarly hoped that EXDAMS will stimulate interest in the closely related but largely neglected problem of monitoring a program by providing new ways of processing the program's behavioral information and presenting it to the user. Or, as a famous philosopher once almost said, "Give me a suitable debugging environment and a tool-building facility powerful (and simple) enough, and I will debug the world."

Design goals

EXDAMS was designed to satisfy three needs: first, as a vehicle to test some proposed, but unimplemented on-line debugging and monitoring facilities; second, as an extendable facility to which new debugging and

* This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0141. Any views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of ARPA.

monitoring aids could be added easily, then tested; and, third, as a system providing some measure of independence of not only the particular machine on which it is being run and the particular implementation of the language being debugged and/or monitored, but also of several source languages in which users' programs could be written and debugged and/or monitored.

The normal techniques for on-line debugging, involving dynamic manipulation of a running program, were inappropriate for these three ambitious design goals, because, first, these techniques were both implementation-dependent and difficult to control; and, second, certain important facilities, such as the ability to run the programs backwards, are impossible with these techniques.

Therefore, the program to be debugged will run with an EXDAMS routine that will monitor it, collect necessary information about the program's actions, and store this information on a *history tape*. Subsequently, EXDAMS debugging routines can retrieve any information from the history tape, format it, and present it to the user. Thus, assuming the history tape is complete (i.e., contains all relevant data), *any* debugging and/or monitoring tool involves only retrieving, then formatting, data from this static file.

The parts of EXDAMS which analyze the program and collect its history (the program-analysis and history gathering phases discussed in a later section) are language dependent. However, the major portion of EXDAMS, and the portion chosen for experimentation—the debugging and monitoring routines—interact with only the history file. They are therefore independent of both the implementation of the source language and the source language itself—to the extent the history file is independent of the differences between source languages, as it is for the common algebraic languages (PL/I, ALGOL, FORTRAN, etc.).

With this approach, the three design goals have been achieved. Any debugging and monitoring aids can be added to EXDAMS easily by writing the appropriate file-search and formatting routines. Moreover, these aids are independent of the implementation of the source language and, to a certain extent, of the source language itself.

However, efficiency has been sacrificed. This approach is based on the insulation from the running program that results from the production of a history tape of the program's behavior. The production and replaying of this history involves large amounts of I/O. However, the flexibility gained far outweighs the inefficiency introduced, especially when one is studying alternative debugging and monitoring aids.

The EXDAMS system output device is a cathode ray tube (CRT) display, and all the debugging and monitoring aids utilize its two-dimensional and high-data-rate capabilities. Some aids, in addition, use the CRT's true graphic (point and vector) and dynamic (time-variant) capabilities. The input devices are a keyboard and some type of graphical pointing device, e.g., a light-pen, RAND Tablet, joy-stick, mouse, or keyboard cursor.

Before describing how EXDAMS works and how new debugging and monitoring aids are added to the system, we present some of the aids currently being added to the basic EXDAMS system to give the reader a better understanding of the types of debugging and monitoring aids that are possible in EXDAMS.

Debugging and monitoring aids within EXDAMS

EXDAMS contains two types of debugging and monitoring aids—static and motion-picture. The static aids display information that is invariant with execution time (a time value incremented as each source statement is executed and used to refer to particular points in the execution of a program), such as the values of variables at the time an error occurred, a list of all values of a variable up to a given execution time, or a display of a portion of the source code.

The motion-picture aids, on the other hand, are execution-time sensitive; that is, the data they display may vary with execution time. These motion-picture aids include the last n values of a set of variables, the current instruction and subroutine, and the current values of a set of variables. The user can run motion-picture aids either forwards or backwards at variable speeds, by controlling execution time.

EXDAMS' most attractive features, from the user's standpoint, are (a) his ability to control his program's execution time, moving at variable speed either forwards or backwards, while a debugging and/or monitoring aid constantly updates its display of information; and (b) his ability to stop execution time at any point, switch to another aid, and continue perusing the behavior of his program.

Static displays

Error analysis

The user requests the value of certain variables at the time an error occurred. The system displays the value of these variables and all other variables in the error-causing source language instruction, the type of error, and the source instruction in error.

Source code

A portion of the user's source code is displayed in optional formats that may include indications of the number of executions per statement and the removal of levels in the source code (such as the bodies of do-groups or the code in the THEN or ELSE clauses) below a certain depth, to afford the user a broader view of his program.

The user may request this display in two manners. He may call for the code around a certain label by requesting SOURCE AT and specifying a label, or a label plus or minus some number of source statements. He also may call for, at any time, the source code around the exact statement that caused a particular value of a variable by requesting SOURCE FOR and specifying the desired value (the source statement causing that value will be marked by its brightness). That is, EXDAMS can associate any value with the exact source statement that produced it.

This ability, and its inverse of associating any source statement with the values it produces, is fundamental to the EXDAMS philosophy of debugging and monitoring that the activity of a program may be viewed in either the data or the control spaces. The data space shows *which* manipulations a program performs, which values change, and the sequence in which they change. The control space demonstrates *how* a program performs its manipulation.

In a canonic debugging situation, according to the EXDAMS philosophy, the user first ascertains *what* is happening, then decides whether this behavior is correct, and finally, if it is not correct, determines *how* the program performed these operations, at the same time seeking the error in the program and/or data. Thus, any comprehensive debugging and monitoring system must include powerful facilities in both the data and the control spaces and provide a simple means of alternating between corresponding points in either space, as the user's needs or personal preferences dictate.

Flowback analysis

By calling for FLOWBACK FOR and specifying a particular value, the user requests EXDAMS to analyze how information flowed through his program to produce the specified value. This analysis appears in the form of an inverted tree, with the bottom node corresponding to the value for which the flowback analysis was desired. Each node consists of the source-language assignment statement that produced the value, the value itself, and links to nodes at the next level. These nodes correspond to the non-constant values in the assignment statement displayed in the node that links with these nodes. These nodes have the same format as the original and are linked to nodes for all non-constant values used in the particular assignment statement producing their value. Thus, Figure 1 shows a flowback analysis for a particular value of A.

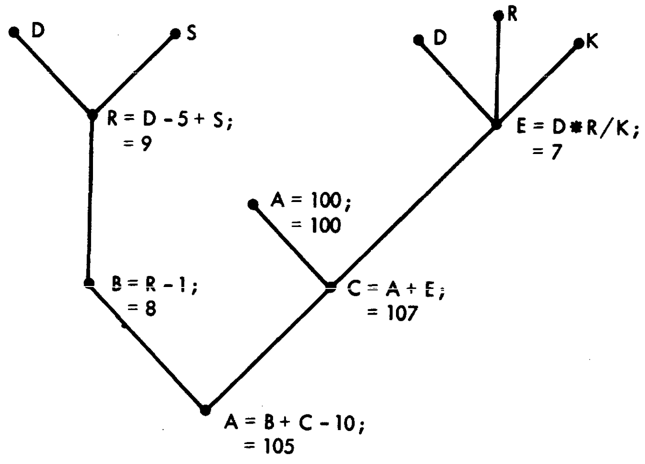


Figure 1

This display shows that the assignment statement "A=B+C–10;" produced the specified value of A, and its value here was 105. The values of B and C used in this assignment to A were 8 and 107, respectively, and were produced by the assignment statements "B=R–1;" and "C=A+E;", respectively. Each of the other nodes is explained in the same manner.

As many levels as will conveniently fit on the screen will be displayed. The user can request a similar flowback analysis along any particular branch. He can also call for the source code around any assignment statement in the flowback analysis and, as described in the section *Motion-Picture Displays* below, watch the execution either forwards or backwards from any point.

A similar type of flowback analysis is possible for the control space, which displays the flow of control through the program between any two points in execution time (i.e., between two nodes in the flowback analysis). In a non-parallel processing environment, this is simply a linear sequence, unless one wishes to indicate control sequences at a lower level (within a subroutine or do-group) as a closed loop out of the main flow of control.

Motion-picture displays

In all the following examples, the information displayed is a function of execution time, whose rate of change the user may increase or decrease, stop, or reverse. Such control, together with the ability to alternate between different debugging and monitoring aids, enables him to discover and pinpoint the bugs in his program.

Values

This facility displays the values of the variables or labels specified by the user. Each specified variable or label is assigned a contiguous set of columns on the display in which their values will appear. (The label values will be a checkmark indicating at what point in the execution the label was reached.) These values will be ordered according to execution time, so that a value produced earlier than another will appear higher on the screen (Figure 2). This display can be scrolled up or down to show other values that cannot fit on the screen at the same time. This scrolling alters execution time appropriately. The user can change the direction of scrolling (and execution time) or stop at any point. Once stopped, he may alter the list of variables on the screen and restart, or he may request the source code for a particular value displayed.

```
ABC      R1         S         FILE      LABEL2

12

                    0

                              FILE1

-10

                              JOE

         '101'B
         '0'B

                    3001

1000

                                        ✓

                              HAL

         '1'B

                                        ✓

1000
```

Figure 2

Source code

This facility allows the user to watch his program statements execute either forwards or backwards. The statement being executed will appear brightened on the screen. If it is an assignment statement, the value of the assignment will also be displayed. If the instruction being executed is not on the screen, the portion of the program containing this instruction will be displayed. The user can command the system to follow subroutine calls and, as in the static display of source code, to display all levels.

Map

This facility is an extension of the source-code facility and is an adaptation of Stockham's work on flow analysis.[3] The user specifies nodes (labels) to be displayed. All code between these nodes may be considered a single macro statement, for the purposes of execution-time advancement. Thus, as the user varies the execution time, the node corresponding to the code being executed brightens and, as execution moves from one node to another, a displayed arrow indicates this shift. The length of time a node brightens is determined by either a common execution-time rate for *each* macro statement or by the execution-time rate for *all* statements executed within the macro statement.

The former display is most useful for following program execution while searching for a bug, while the latter is well-suited to monitoring applications in which the user is trying to determine how the program operates and where it spends most of its time. The EXDAMS environment—allowing the user to dynamically stop the display, expand some nodes into several separate nodes, collapse other nodes into a single node, and then continue or reverse direction—should greatly improve the usefulness of this display.

Windows

The current values of the variables specified appear in "windows" (i.e., areas on the display screen) as execution advances or reverses. If the value exceeds the size of the window, as much will be displayed as possible. In the case of arrays, the system will display in the window the value being changed and as many array elements, and their indices, around it as can fit. In addition, for either arrays or strings, certain variables can be specified as pointers or indices into these data representations.

The values of these variables appear in graphic, rather than numeric or alphanumeric, form according to the position of an arrow directed at the character or element at which the pointer or index is also directed.

Thus, in a buffer application, where many buffers are scanned and processed and new buffers created, the user can watch the data in the buffers change dynamically, and see the pointers and indices move back and forth through the buffers.

Windows with transitions

This facility performs the same operations as the preceding *Windows* facility except that, in addition, it indicates the interrelationships between the displayed variables. As each new value is displayed, a flowback analysis determines whether the current value of any displayed variable was used in the creation of the new value. If so, an arrow indicating this dependence appears, linking the windows of these variables to the window of the variable being changed. To obtain more detail for a particular transition indicated by the arrows, the user may define a new display relevant to that transition only, then either re-advance or reverse execution time. After completing the study of this particular transition, he may return to his original display.

The EXDAMS environment

EXDAMS is a four-phase system predicated on the assumption that neither an incremental compiler nor a special debugging compiler designed for EXDAMS requirements would be available for the source language being debugged and/or monitored. If either is available, considerable restructuring of these phases would be prerequisite to the full utilization of these capabilities. The four phases are program analysis, compilation, run-time history-gathering, and debug-time history-playback.

Program analysis

The first phase analyzes the user's source program as it performs four functions, the most important of which is the creation of a model of that program. This model, the heart of the debug-time history-playback, is the means by which values gathered on the history tape are interpreted and by which portions of the source code are retrieved, and is the repository of all structural information known about the program. The use of the model for these functions will be explained in the section *Debug-Time History Playback* but the contents of the model will be discussed here.

The program analysis produces both a symbol table and a random-access file of the user's source program for the history-playback. As it analyzes the program, it also builds a model of the program, and inserts debugging statements into the program to provide the information necessary for history-gathering. In general, the history contains all the dynamic information needed to update execution time either forwards or backwards, while the model contains all necessary static information.

Each model entry consists of an indicator of the type of model entry, a pointer to the associated source statement, and an index to an entry in either the model or the symbol table, depending on the type of entry.*

The model contains both the static control-information and the variable alteration-information of the user's program. The control-information consists of the CALL, GOTO, IF-THEN-ELSE, and DO-END structure of the program, while the variable alteration-information consists of the names of the variables on the left-hand side of assignment statements and those altered by input statements.

The debugging statements added to the program pass the relevant run-time information to the run-time history-gathering routines.**

The updated program is passed to the compilation phase, while the symbol table and model are saved for the debug-time history-playback.

Compilation

The standard source-language processor compiles the source program, as updated during program analysis.

Run-time history-gathering

The compiled version of the updated program is run with a set of run-time routines that it calls. These routines gather dynamic information about the program's behavior. This information is collected in a buffer that is written out when full. It is the *history tape* of the programs's behavior and, together with the symbol table and model, is sufficient to recreate the program's behavior in either the forwards or backwards direction of execution time. This history contains, basically, the values of the variables on the left-hand side of assignment statements, the direction (THEN or ELSE) taken in IF statements, the direction (remain in or flow out) taken at the end of DO-LOOPS, and the point from which a GOTO or CALL was issued (to facilitate execution-time backup).†

* Appendix A explains the use of the index field for each type of model entry.

** Appendix B details these statements.

† Appendix C presents the precise information placed in the history.

Debug-time history-playback

This phase contains the debugging and monitoring aids which present the history information to the user in a usable form on his display screen. It also interprets the user's commands for alternative displays and/or execution-time variations, and provides an editing capability for modifying discovered bugs and for returning this modified program to the four phases for another debugging iteration.

The main function of this phase is to assemble information from the history and display it on the screen. Appropriately, the main routine in the phase is the information retriever used by all the debugging and monitoring aids to retrieve desired information from the history. It accepts (a) requests from the processing routines for information on a certain variable or set of variables and (b) a direction for execution-time. Using this direction, it searches the history for the next occurrence of a value change for any variable in the requested set. It returns the name of this variable, its new (or old, if executing backwards) value, and its attribute.

Special calls facilitate the next subroutine call, goto, return, assignment, iteration, or conditional statement to be retrieved, so that all information in the history is retrievable through this routine. The calling routine describes what information to retrieve, and combines, processes, and formats it for the display routines that interact with the display equipment.

The information retriever moves a marker through the model as values are read in from the history. This movement serves three purposes:

1. To permit interpretation of the bits in the history. Since the values in the history are not of a fixed length, knowledge of the type of the next value allows the routine to correctly interpret the value and position itself at the next value.
2. To associate the values in the history with statements in the source program (through the pointer to the source statement in the model), enabling users to alternate between values in the data space and the associated source statements in the control (program) space.
3. To reduce the amount of I/O necessary. By using the model to interpret values from the history, we need store only the value of source variables and not also the identification of the variables of which it is the value. This reduces the amount of I/O by roughly one-half; since the system is I/O-bound, this obviously improves the system's response.

The addition of new debugging and/or monitoring facilities

To add a new debugging and/or monitoring facility to the EXDAMS system, first, extend the command language of EXDAMS to include the new commands needed to control the new facility and to route control to the new routine for these commands. As long as these commands do not conflict with existing ones, this is an easy task.

Second, obtain the information required to respond to the new commands by requesting it from the information retrieval routine as described in the previous section. This is the essential issue in the EXDAMS philsophy: All the information required by a routine can be obtained easily, by request, without interacting with the source program, the object code, or the history, but only with the information retrieval routine.

Third, process and combine the obtained information. The ease or difficulty depends entirely on the facility being added and is independent of the information collection mechanism.

Finally, format and display the processed information. Again, the effort required depends entirely on the facility being added and is independent of the monitoring mechanisms.

Thus, the EXDAMS environment reduces the problems of collecting information for a debugging and/or monitoring facility, but provides only minimal capabilities in the processing and presentation of this information. If the collection of information is a major problem in the creation of a debugging and/or monitoring facility, then EXDAMS has met its design goals. In addition, as we gain more experience in the types of processing and formatting required, we may also be able to provide capabilities that facilitate these areas.

Example

To illustrate the EXDAMS system, we present an example source program written in PL/I,[4] followed by the major transformations performed on it by EXDAMS.

Original source program *

```
1. example program: PROCEDURE OPTIONS
      (MAIN);
2.    DECLARE
3.       a (10, 3) CHARACTER (8) EXTERNAL,
```

* The reserved keywords[4] of the source language are in all capital letters.

```
     i BINARY FIXED,
     switch BIT (1),
     search_string CHARACTER (8) VARYING;


   GET FILE (input) LIST (switch, search_string);
   IF switch THEN
loop: DO i = 1 TO 10;
     DO j = 1 TO 3;
       IF  a(i,  j) = search_string  THEN  DO;
         PUT LIST (i, i < j);
         CALL abc (i, i + j * 3);
         GO TO end_program;
         END;
     END loop;
   ELSE
   PUT LIST ('switch turned off');
end_program:
   i = j * i - 5;
   RETURN;
   END example_program;
```

Symbol table

The data are formatted here to facilitate reading, but this format does not reflect actual internal representation. The dummy entries (12 through 17) at end of the Symbol Table represent the types of expressions being passed to a subroutine or output.

Symbol Number	Name	Attributes	Model Entry Number
1	A	ARRAY (*, *), CHARACTER (8)	
2	ABC	PROCEDURE (*, *)	
3	END_PROGRAM	LABEL	26
4	EXAMPLE_PROGRAM	PROCEDURE	1
5	I	BINARY, FIXED	
6	INPUT	FILE, STREAM	
7	J	BINARY, FIXED	
8	LOOP	LABEL	6
9	SEARCH STRING	CHARACTER (8), VARYING	
10	SWITCH	BIT (1)	
11	SYSPRINT	FILE, STREAM	
12	DUMMY	BINARY, FIXED	
13	DUMMY	DECIMAL, FIXED	
14	DUMMY	BINARY, FLOAT	
15	DUMMY	DECIMAL, FLOAT	
16	DUMMY	CHARACTER (*), VARYING	
17	DUMMY	BIT (*), VARYING	

Model

To facilitate the reader's interpretation, the pointer to the source code is represented here as a line number in the original program, and an explanation of the index field of the model entry has been added.

Model Entry Number	Entry Type	Source Code Pointer	Index to Model or Symbol Table	Explanation of Index
1	PROCEDURE	1	29	Index of associated END model entry.
2	GET	9	6	Index of Symbol Table of file associated with GET.
3	GET_ASSIGNMENT	9	10	Index of symbol receiving new value.
4	GET_ASSIGNMENT	9	9	Index of symbol receiving new value.
5	IF	10	22	Index of model entry for end of THEN clause.
6	LABEL	11	8	Index of label in Symbol Table.
7	ITERATIVE_DO	11	21	Index of model entry for associated END statement.
8	ITERATIVE_ASSIGNMENT	11	5	Index in Symbol Table of iteration variable.
9	ITERATIVE_DO	12	20	Index of model entry for associated END statement.
10	ITERATIVE_ASSIGNMENT	12	7	Index in Symbol Table of iteration variable.
11	IF	13	19	Index of model entry for end of THEN clause.

(Notice there is no entry for the non-iterative DO statement in line 13 of the source code.)

12	PUT	14	11	Index of Symbol Table entry of file receiving new value.
13	PUT_ASSIGNMENT	14	5	Index in Symbol Table of first output value.
14	PUT_ASSIGNMENT	14	17	Index in Symbol Table of second output value. (This is a dummy entry for the attributes (bit) of the output expression.)
15	CALL	15	2	Index of label in Symbol Table.
16	CALL_PARAMETER	15	5	Index in Symbol Table of value being passed as first parameter.
17	CALL_PARAMETER	15	12	Index in Symbol Table of value being passed as second parameter. (This is a dummy entry in Symbol Table that represents the attributes of the expression being passed.)
18	GOTO	16	3	Index of label in Symbol Table.
19	SHORT_IF_END	17	11	Index of model entry of associated IF statement.
20	ITERATIVE_END	18	9	Index of model entry of associated iterative do.
21	ITERATIVE_END	18	7	Index of model entry of associated iterative do.
22	ELSE	19	25	Index of model entry of end of ELSE clause.

Model Entry Number	Entry Type	Source Code Pointer	Index to Model or Symbol Table	Explanation of Index
23	PUT	20	11	Index in Symbol Table of file receiving new value.
24	PUT_ASSIGNMENT	20	16	Index in Symbol Table of first output value. (This is a dummy entry for the attributes (character) of the output expression.)
25	FULL_IF_END	20	5	Index of model entry of associated IF statement.
26	LABEL	21	3	Index of label in symbol table.
27	ASSIGNMENT	22	5	Index in Symbol Table of variable left of assignment statement.
28	RETURN	23	4	Index of associated procedure label in symbol table.
29	PROCEDURE-END	24	1	Index of model entry of associated procedure statement.

Augmented source program

The altered or inserted source statements are italicized to facilitate their recognition.

```
example_program:  PROCEDURE OPTIONS (MAIN);
     DECLARE
          a (10,5) CHARACTER (8) EXTERNAL,
          i BINARY FIXED,
          switch BIT (1),
          search_string CHARACTER (8) VARYING;
     DECLARE condition_tester RETURNS (bit 1));
     GET FILE (input) LIST (switch, search_string);
     CALL bit_value (switch); /*record new value*/
     CALL character_value (search_string); /*record
       new value*/
     IF condition_tester (switch) THEN DO; /*record
       valu e of if condition*/
          CALL goto_issued_from (5); /*record index
            of model entry from which control passed
            to label*/
loop:
          CALL outside_do_loop; /*record control outside
            of do-loop*/
          DO i=1 to 10;
                CALL inside_do_loop; /*record control
                  inside of do-loop*/
                CALL binary_fixed_value (i); /*record
                  new value*/
                CALL outside_do_loop; /*record control
                  outside of do-loop*/
                DO j=1 TO 3;
                CALL inside_do_loop; /*record control
                  inside of do-loop*/
                CALL binary_fixed_value (j); /*record
                  new value*/
                     IF condition_tester (a(i,j)=search_string)
                       THEN DO; /*record value of if condition*/
                          PUT LIST (i,i<j);
                          CALL binary_fixed_value (i); /*record
                            output value*/
                          CALL bit_value (i<j); /*record output
                            value*/
                          CALL called_from (15); /*record index
                            of model entry of call statement*/
                          CALL binary_fixed_value (i); /*record
                            value of passed parameter*/
                          CALL binary_fixed_value (i+j*3);
                            /*record value of passed parameter*/
```

```
                    CALL abc (i,i+j*3);
                    CALL goto_issued_from (18); /*record
                      index of model entry of goto
                      statement*/
                    GOTO end_program;
                    CALL end_then_clause; /*record end
                      of then clause*/
                    CALL inside_do_loop; /*record control
                      at end of do loop*/
                    END;  /*explicitly end each iterative
                      do loop*/
               CALL outside_do_loop; /*record control
                 outside of do-loop*/
               CALL inside_do_loop; /*record control at
                 end of do-loop*/
               END loop;
          CALL outside_do_loop; /*record control outside
            of do-loop*/
          CALL end_then_clause; /*record end of then
            clause*/
          END; /*end non-iterative do group inserted
            after if statement*/
     ELSE DO; /*add do to enclose added statements
       within else clause*/
          PUT LIST ('switch turned off');
          CALL character_value ('switch turned off');
            /*record output value*/
          CALL end_else_clause; /*record end of else
            clause*/
          END; /*end non-iterative do group inserted
            after else statement*/
     CALL goto_issued_from (25); /* record index of
       model entry from which control passed to label*/
end_program:
     i=j*i-5;
     CALL binary_fixed_value (i); /*record new value*/
     CALL return_issued_from (28); /*record index of model
       entry of return statement*/
     RETURN;
     END example_program;
```

History file

We assume the input value for switch and search string to be TRUE and 'XYZ', respectively. We further assume that the first element of array A that matches this string is A(2, 1). The format of the values in the file facilitates reading. Comments appear on the right. The reader can start either at the end of the history and work backwards to the beginning of the program, or at the beginning of the history and work forward towards the end of the program.

Value	*Comments*
TRUE	Input value of SWITCH.
XYZ	Input value of SEARCH_STRING.
TRUE	Value of if-condition.
5	Index of model entry from which goto was issued.
OUTSIDE_DO_LOOP	Control is outside outer do-loop.
INSIDE_DO_LOOP	Control is inside outer do-loop.
1	Value for iteration variable I.
OUTSIDE_DO_LOOP	Control is outside inner do-loop.
INSIDE_DO_LOOP	Control is inside inner do-loop.
1	Value for iteration variable J.
FALSE	Value of if-condition.
INSIDE_DO_LOOP	Control is at end of inner do-loop.
INSIDE_DO_LOOP	Control is at beginning of inner do-loop.
2	Value for iteration variable J.
FALSE	Value of if-condition.
INSIDE_DO_LOOP	Control is at end of inner do-loop.
INSIDE_DO_LOOP	Control is at beginning of inner do-loop.
3	Value for iteration variable J.
FALSE	Value of if-condition.
INSIDE_DO_LOOP	Control is at end of inner do-loop.
OUTSIDE_DO_LOOP	Control is outside inner do-loop.
INSIDE_DO_LOOP	Control is at end of outer do-loop.
INSIDE_DO_LOOP	Control is at beginning of outer do-loop.
2	Value for iteration variable I.
OUTSIDE_DO_LOOP	Control is outside inner do-loop.
INSIDE_DO_LOOP	Control is inside inner do-loop.
1	Value for iteration variable J.
TRUE	Value of if-condition.
2	Output value on file SYSPRINT.
FALSE	Output value on file SYSPRINT.
15	Index of model entry from which call was issued.
2	Value of parameter being passed.
5	Value of parameter being passed.
18	Index of model entry from which goto was issued.
−3	New value for I.
28	Index of model entry from which return was issued.

REFERENCES

1 F J CORBATO V A VYSSOTSKY
Introduction and overview of the multics system
Proc F J C C Spartan Books, Washington,D.C.
1965 185–196

2 T G EVANS D L DARLEY
On-line debugging techniques: A survey
Proc F J C C Spartan Books Wasington D.C.
1966 37–50

3 T G STOCKHAM JR
Some methods of graphical debugging
Proc IBM Scientific Computing Symposium on Man-Machine Communications Thomas J. Watson Resarch Center Yorktown Heights N Y
May 3–5 1965

4 *IBM operating system/360 PL/I: Language specifications*
IBM Corporation White Plains N Y Form C28–6571–4
1967

5 H FERGUSON E BERNER
Debugging systems at the source language level
C ACM Vol 6 No 8 August 1963 430–432

6 M HALPERN
Computer programming: The debugging epoch opens
Computers and Automation Vol 14 No 11 November 1965
28–31

APPENDIX A

Use of index field in model entries

Model Entry	*Use of Index Field*
PROCEDURE, BEGIN, ITERATIVE_DO,	Index of associated END model entry.
DO_WHILE	
END	Index of associated PROCEDURE BEGIN, ITERATIVE_DO, or DO_WHILE model entry.
IF	Index of associated ELSE (if this is an IF-THEN-ELSE s tatement) or SHORT_IF_END (if this is an IF-THEN statement) model entry.
ELSE	Index of associated FULL IF END model entry.
SHORT_IF_END, FULL_IF_END	Index of associated IF model entry.
CALL, FUNCTION_INVOCATION, GOTO, LABEL	Index in Symbol table of associated label.
RETURN	Index in Symbol table of associated Procedure label.
GET, PUT	Index in Symbol table of associated file.
ASSIGNMENT, GET_ASSIGNMENT, PUT_ASSIGNMENT, ITERATIVE ASSIGNMENT, CALL_ARGUMENT	Index in Symbol table of associated variable (or dummy entry if an expression).

APPENDIX B

Statements added to user program to produce EXDAMS history

In the additions described below it is assumed that DO-END brackets are placed around statements as necessary to preserve the semantics of the user program, e.g., when the THEN clause is expanded from one statement to two or more.

1. For each variable on the lefthand side of an assignment statement, each parameter in a function or procedure call, and each variable in an input or output statement, a call to the appro-) priate (as determined by the item's attributese value saving routine, passing the item as th. argument, is inserted after the source statment In addition, for each parameter in a function or procedure call these same value saving calls are also inserted before the source statement.
2. For each IF statement the condition is replaced by a function call (which saves the value of the condition) with the condition as the argument of the function call.
3. At the end of a THEN clause a call is made to the END_THEN_CLAUSE routine.
4. Similarly at the end of an ELSE clause a call is made to the END_ELSE_CLAUSE routine.
5. Before a CALL statement, a RETURN statement, or a GOTO statement or the occurrence of a label a call to the CALL_ISSUED_FROM, RETURN_ISSUED_FROM, or GOTO_ISSUED_FROM routine (passing the entry number of the associated model entry as the argument) is inserted.
6. Before an ITERATIVE_DO or DO_WHILE statement a call to OUTSIDE_DO_LOOP is inserted. After the source statement a call to INSIDE_DO_LOOP is inserted. In addition, if the source statement is an ITERATIVE_DO statement, a call to the appropriate value saving routine (passing the control variable as argument) is inserted after the call to INSIDE_DO_LOOP.
7. Before an END statement which is an end to an ITERATIVE_DO or DO_WHILE statement a call to INSIDE_DO_LOOP is inserted, and after the END statement a call to OUTSIDE DO_LOOP is inserted. If the END statement specifies a label, it is replaced by the appropriate number of simple END statements before the above additions are made.

APPENDIX C

Information recorded in history for each type of statement

Statement Type	*Information Recorded in History*
1. Assignment	Value of each variable on left hand side of assignment statement after the assignment is made.
2. IF	Value of if-condition.
3. end of then clause	Indication of end of then clause.
4. end of else clause	Indication of end of else clause.
5. Call, function invocation goto, and return statements	Index of model entry associated with source statement, and for return statement, index of model entry to which return is being made. In addition, the value of each argument in the call or function invocation is saved both before and after the call or function invocation.
6. iterative_do, do_while	Indication of the start and end of do_loop and two indicators signalling each iteration around the loop. Also, for iterative_do loops, the value of the control variable is saved each time around the loop.
7. input, output	Value of each variable (or expression) for which a value was input or output after the input or output operation.

Maximum-rate pipeline systems

by L. W. COTTEN

National Security Agency
Fort Meade, Maryland

INTRODUCTION

There is widespread opinion that we are fast approaching the physical limit in speeds for computers. The grounds for such conclusions are traceable to signal propagation delays in interconnections, delays encountered in traversing several levels of combinatorial logic, and to systems organizations. Clearly, these areas must be addressed if we are to realize phenomenal improvements in computer logic speeds over the next decade. Subnanosecond logic circuits will be available; however, design innovations are needed to exploit this performance at the systems level.

Trends in the organization of current high-performance systems (Parallel bits $\times$ clock rate $\geq 10^9$ bits/sec.) have tended toward three types of parallelism. These are arrays of processor elements, processing of operand bits in parallel, and pipelining[1–6] or compaction of operations in the time domain. In the processing of blocked operands or data arrays[7,8] all three are potentially complementary rather than competitive. Since most large problems permit iterative processing of blocks of operands, pipelining becomes attractive because the rate of doing work can be increased considerably without a corresponding linear increase in hardware cost. The pipeline queue is the result of organizing logic into an assembly-line process where output rate is independent of the latency or total delay of the pipeline. At any given time a number of operands are in various stages of processing, thus the additional hardware usually takes the form of temporary storage and process control.

The purpose of this paper is to explore the problem of maximum-rate pipeline operation from a viewpoint that differs markedly from current practice. The results will be termed maximum-rate pipelines to distinguish from conventional pipeline systems. In truth, conventional pipelines as presently designed are a special case of the maximum-rate pipeline which constitutes the general case. To clarify, all pipeline systems in existence today are based on a rate which is the reciprocal of the delay through an elemental pipeline section, typically consisting of a parallel register and perhaps three levels of logic. In contrast this paper proposes a rate of operation largely dependent on the delay difference of the various logic paths through the pipeline section. Since this difference delay can be made considerably less than total delay, higher theoretical rates are possible. Indeed, such rates are suprisingly independent of both propagation delays through N-1 levels of gating between clocks, and pure signal transmission delays caused by distances between gates. As will be shown, the maximum rates are limited by uncertainty or variability of paths, and basic circuit switching rate. The latter limit has long been recognized.[9] It is hoped that the following treatment will provide helpful insight and some design directions that will prove useful in overcoming some of the problems[10,11] in the high-performance environment.

As a guide to readers, the next two sub-topics will discuss conceptual aspects of maximum-rate pipelines, while the remaining sub-topics will consider the complexities of implementation. The latter subjects may be selectively examined depending upon individual interests. While 50 MHz floating point arithmetic units have been designed using the conventional pipeline concept, no maximum-rate design results are available; however, a numerical example to aid understanding is given at the end of the discussion on logic level variability.

A maximum-rate pipeline queue

The validity of maximum-rate pipeline systems, as with pipeline systems in general, is dependent upon the ability of designers to relate data processing problems to an assembly-line job queue. Feedback is sometimes permitted to slightly reduce average rate, but in gereral can usually be deferred in time at hardware expense. For example, this problem has been investigated in the case of accumulators.[12] Since pipeline systems designers must demonstrate the ability to resolve out feedback dependencies, and multiple path interactions will be considered later, it would appear that attention could now turn to the case of a unidirectional pipeline. In the interest of brevity and intuitive understanding, a physical analog of an elementary queue offers some utility.

Consider the conventional pipeline, Figure 1, case 1. Distance X is defined to be interstation separation, and does not include the finite station width. The propagation velocity V between stations is assumed to be positive in the direction shown and non varying. S is defined to be service time at each station, and includes the time needed to traverse the station width. Service time S is analogous to data sampling and temporary storage in logic pipelines, and X/V corresponds to propagation delay through logic circuits and interconnections between temporary storage latches, as seen in Figure 2. Under these conditions throughput rate is:

$$R_1 = 1/\ (S + X/V) \qquad \text{units/second} \qquad (1)$$

The rate R_1 summarizes the present level of sophistication in pipeline machine design. Borrowing heavily from the notion of a communications channel containing many information symbols or bits in transit, one is enabled to visualize the idealized maximum-rate pipeline, Figure 1, case 2. If S includes time consumed in traversing the finite station width, then ideal rate R_2 is:

$$R_2 = 1/S \qquad \text{units/second} \qquad (2)$$

Ideal rate R_2 is impossible to realize in practice, because of symbol interference caused by path eccentricities. In theory, the rate approaches 1/S as eccentricity is made to approach zero.

In practical circumstances the eccentricity is usually predictable within acceptable bounds, and can be compensated for in Figure 1, case 3, by the addition of a separation ΔX. Under assumptions that the receiving station can undergo minor repositioning or phasing relative to the sending station, the rate becomes:

$$R_3 = 1/\ (S + \Delta X/V) \qquad \text{units/second} \qquad (3)$$

If V is a universal constant, the velocity of light for example, its effect could be minimized by reducing the quantity $\Delta X/V$, where ΔX is intersymbol separation as distinguished from X, interstation separation. The time S in (3) is equivalent to the width of one Nyquist interval,*[13,14] and 1/S corresponds to the rate of completeness property which represents the maximum input sequence rate for which any finite state machine can be built.[15,16]

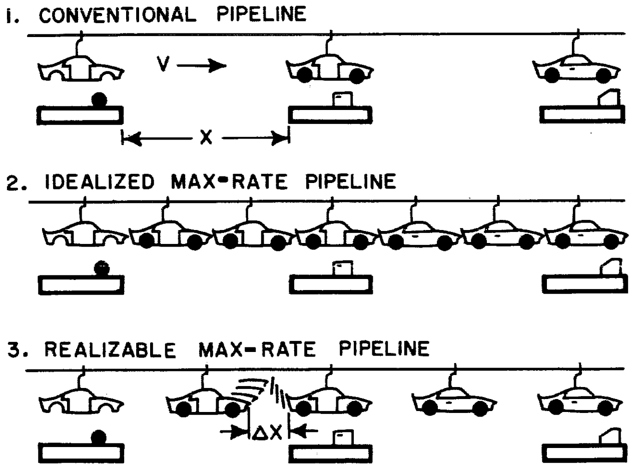

1. CONVENTIONAL PIPELINE
V
X
2. IDEALIZED MAX-RATE PIPELINE
3. REALIZABLE MAX-RATE PIPELINE
ΔX

Figure 1—Pipeline queue examples

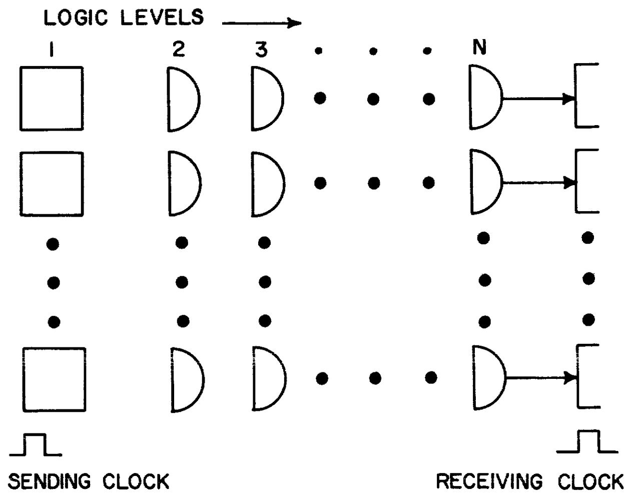


Figure 2—Generalized pipeline section

A maximum-rate pipeline section

The main result of this paper is to apply the max-rate queue concept to a generalized max-rate pipeline section which may be joined with other max-rate or compatible-rate sections to construct max-rate systems. The design of the generalized max-rate section is related to two characteristics of the continuous data stream,

* The Nyquist interval is the minimum useful pulse width resolvable by logic gates.

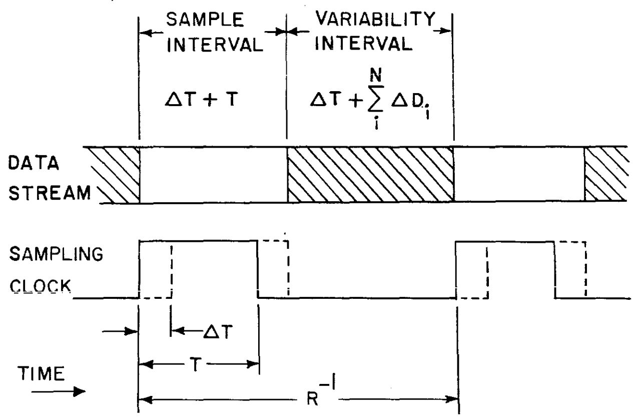


Figure 3—Continuous data stream

Figure 3. The sample interval is a function of the fixed clock-pulse width, defined to be T, and clock skew (ΔT). Skew ΔT is the total range of variability in arrival time of the clock pulse, as observed at the clocked latch. The clock could be present as late as time T + ΔT, therefore the sample interval width is:

$$\text{Sample} = T + \Delta T \qquad (4)$$

The variability interval is defined as the time interval over which one discrete data arrival occurs, but whose exact arrival time cannot be predicted, and half the time not observable because it is indistinguishable from the preceding state. The variability interval, observed at the receiving latch input, is a function of a composite combinatorial path leading from the sending latch clock. The variability additively builds up over path delay (P) through a pipeline section composed of N logic circuit levels, each with level delay Di, such that

$$\text{Variability} = (\Delta T + P_{\max}) - P_{\min}$$

$$= (\Delta T + \sum_{i=1}^{N} {}_{\max} D_i) - \sum_{i=1}^{N} {}_{\min} D_i$$

$$= \Delta T + \sum_{i=1}^{N} \widehat{D}_i - \sum_{i=1}^{N} \check{D}_i$$

$$= \Delta T + \sum_{i=1}^{N} (\widehat{D}_i - \check{D}_i)$$

$$= \Delta T + \sum_{i=1}^{N} \Delta D_i \quad ,$$

$$\text{where } \Delta D_i = \widehat{D}_i - \check{D}_i \qquad (5)$$

Combining the results of (4) and (5), the maximum clock rate R is:

$$R = 1/\,(\text{Sample} + \text{Variability}) \quad \text{cycles/second}$$

$$R = 1/\,(T + 2\,\Delta T + \sum_{i=1}^{N} \Delta Di) \quad \text{cycles/second} \quad (6)$$

In practice, T can be made to approach one circuit delay for a one-circuit delay type of latch, Figure 4, and subnanosecond circuit delays are possible.[17] The latch is the basic storage cell used in the sample and store operation. Based on past work[5] reasonable values of ΔT are 10 percent/R.

The most significant conclusion is that a summation of the circuit delay difference at each logic level constitutes the principle term to be minimized for higher rates. The effects of finite signal propagation times are thus reducible to arbitrarily small consequences, since in theory delay could be added to fast paths to minimize delay differences. Stating the conclusion more abstractly, the max-rate philosophy of design advocates minimizing differences in naturally occurring absolute quantities, whereas the classical approach to logic design has always been concerned with minimizing the absolute quantities themselves. Each philosophy will

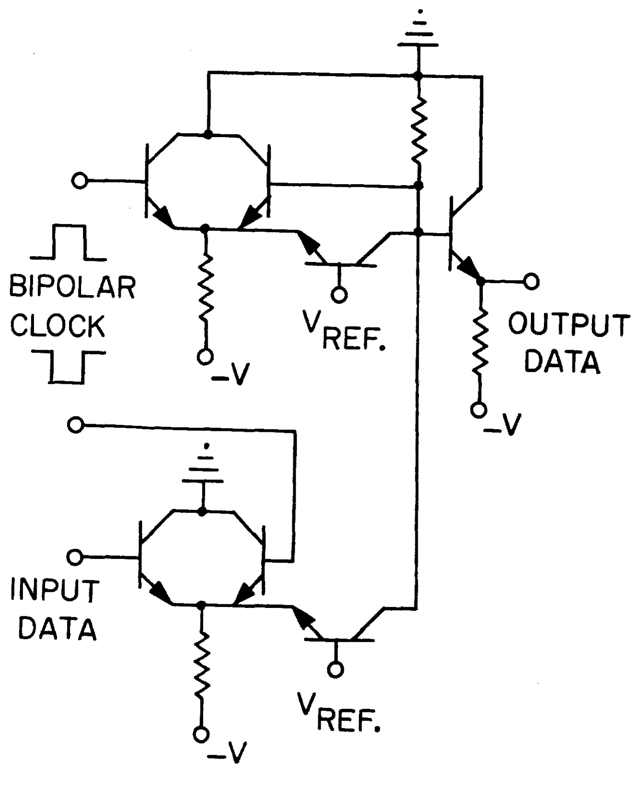


Figure 4—One-delay sampling latch

offer special advantages depending upon technology, systems goals, and acceptable design constraints.

Data rates and logic-path bandwidth

Clock pulse width depends primarily on circuit delay; however, variability is dependent on path bandwidths, which suffer from losses, mismatches, and loading. Theoretical data rates could approach the Nyquist rate, 2f, where f is the abrupt cutoff frequency of an ideal low pass filter, approximated by the logic.[14] The minimum width pulse or bit equals one Nyquist interval, or 1/2f. In the environment a more practical definition of the Nyquist rate is the maximum signaling rate at which intersymbol interference vanishes. This implies full swings, without compromise of noise margins. In practice the Nyquist rate for a string of gates must be lower than that for a single gate. This results from pulse width variation that is not Gaussian in steep-cutoff bandwidth limited systems. These and circuit relaxation effects tend to hasten bit interference, especially for the isolated 1 or Ø bit cases, and could lead to smearing and pulse pinch-off for marginal rates.

In synchronous systems, where variability is accounted for by retiming, the ideal rate would probably never exceed half the Nyquist rate of logic gates. If repetitive clock pulses are funished by a logic gate, the pulse width could equal one Nyquist interval, but pulses would have to be separated by one Nyquist interval. Under these assumptions the maximum realizable rate is:

$$R = 1/\,[2\,(T + \Delta T)] \quad \text{cycles/second} \qquad (7)$$

where:

$\pm\,\Delta T/2$ = Random jitter for the limiting clock case.

$|T + 2\Delta T|$ = Sample interval magnitude

$|T|$ = Variability interval magnitude

The variability interval is taken to be one Nyquist interval for this special case, and would prove too severe for ordinary design purposes. Instead, as provided for in (6), an arbitrarily large variability interval may be accommodated by a corresponding rate reduction. It follows also that (7) establishes an upper bound for (6), assuming the pulse portion of the clock system is implemented with logic gates.

In practice designers empirically arrive at path-bandwidth and analytic approximations, as no present mode is available at this level of complexity. In the past, design relationships have been derived for each new technology and associated effects of loading, transmission line phenomena, and minimum useful pulse width. In addition to past measures of circuit delay, it is important that future max-rate technology be given additional definitions of performance. Of considerable interest are the Nyquist rates for worst-case bit patterns, taken for a continuum of input transition times. This characterization could null out package delay influences that otherwise appear as additive delay in present measurements. The lead and package delay should be treated as wire delay which delays computation, but need not cause a reduction in rate. By similar reasoning a clock pulse width, on the order of a Nyquist interval, should not be widened to account for finite lead-package delay encountered while entering a latch of Lilliputian dimensions compared to the package. A second type of performance characterization should be statistical measures of delay variability to account for production spreads, variable input transition times, and asymmetry in circuit switching characteristics.

Minimizing logic level variability

The throughput data rate approaches the maximum realizable clocked rate (7) when in (6):

$$\sum_{i=1}^{N} \Delta D_i \rightarrow |T| \qquad (8)$$

It is significant that the expression (8) to be minimized is a summation of artificial differences, as contrasted with natural barriers such as signal propagation velocity and combinatorial gate delay through N-1 levels of logic between clocks.

A straightforward variability minimization algorithm is needed if several complex considerations are to be avoided. Statistical averaging through several levels of gating must not be used, as this places complex dependencies on all gates in a section. This is hazardous because in a regular m by n gate array there are m^n paths possible, and these paths could merge $m!n/(m-2)!(2)!$ ways. Any of these mergers, because of delay differences, could have relative bit offsets that might cause symbol interference at an output. One algorithm that avoids this complexity is to make each logic level independent from the others. This can be implemented by permitting designers to reduce $\Delta D_i = \hat{D}_i - \check{D}_i$ on a level by level basis by any means available, but requiring that ultimately a max D_i and min D_i be specified for every level. Designers could tailor each level to achieve maximum speed, or design could be standardized about a few restricted cases to cover all situations.

Wire and loading delay is included at each level. Only the uncontrolled differences in the circuit-line-loading delay triplet need contribute to ΔD_i. Even here, longer path routing or more loading may be used to slow down fast lightly loaded gates. Increases in transition times due to loading are accounted for either in increased transmission line delay or increased delay of driven gates. Parallel line delays such as the six nanosecond delays shown in Figure 5 would not affect the maximum rate as calculated in (6). In fact (6), along with the Figure 5 example data shown in Table I. predicts a 125 MHz clock rate. By contrast, 42 MHZ results if a summation of maximum delays along with clock skew is permitted to determine the period of the clock.

Clock phasing

In order to insure sampling from the nonchanging portions of bit intervals each receiving station clock

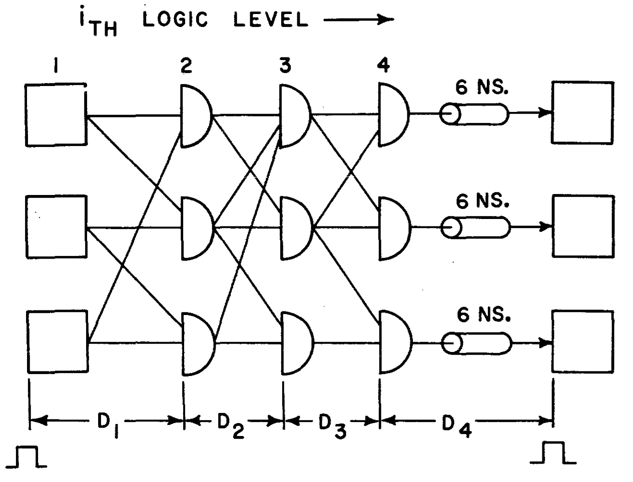


Figure 5—Representative pipeline section

Table I—Example delay data

i	1	2	3	4
$\hat{D}_i$ NS.	4.1	5.9	4.2	3.0 + 6.0
$\check{D}_i$ NS.	3.7	4.8	3.6	2.7 + 6.0
ΔD_i NS.	0.4	1.1	0.6	0.3 + 0
CLOCK WIDTH	T = 4.0 NS.			
CLOCK SKEW	ΔT = 0.8 NS.			

must be offset from the sending station clock an amount:

$$\text{Clock offset} = (\Delta T + \sum_{i=1}^{N} \hat{D}_i) \text{ Modulo } 1/R \quad (9)$$

This resembles a spatially rippled clock which has been mentioned.[18] This requirement could be met by distributing a global time reference, treated as an independent machine,[19] about a 50,000 gate system to within phase differences of say ± 0.2 ns. Next a delay element similar to Figure 6 could be used at each place where a phase is needed, typically for each 150-300 gate section. A stored or wired address would establish the phase to be supplied.

Since all phases were assumed to exist in (6) and actually only p discrete phases are available some reduction in rate becomes necessary. The worst-case rate reduction need never exceed a decrease of 100 percent/p. In reality one would use the a priori knowledge concerning phasing, and constrain the variability interval, thus the reduction in rate is more a theoretical entity. Lastly, the most attractive approach toward realizing some systems is to constrain design such that (9) meets the zero offset case for all sections. This results in a single phase clock, and the requirement for multiple phases no longer exists.

Troubleshooting a max-rate system

Troubleshooting and stepping a maximum-rate system has some new aspects. At any time a dynamic path may contain more bits than storage latches such that storage is not available for the excess bits, or:

$$\text{Excess bits} = R\,(\Delta T + \sum_{i=1}^{N} \hat{D}_i) - 1 \quad (10)$$

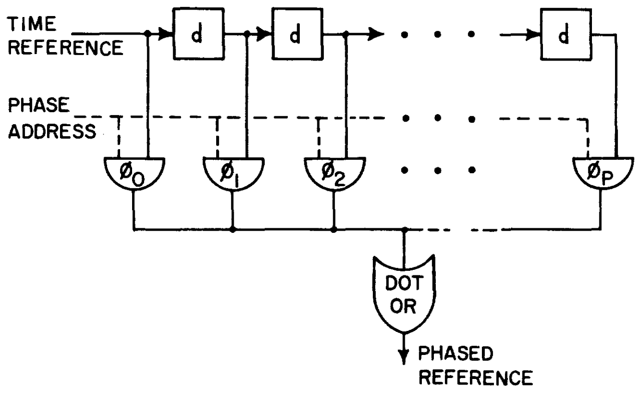


Figure 6—Delay with gated phases

This bears similarity to a delay line. The point to be remembered is that max-rate machines possess the complete personality of a single phase machine, with no excess bits, operating at a lower rate determined by the max-delay section in the system. Also, to be race free in this mode a requirement must be met such that:

$$\sum_{i=1}^{N} \breve{D}_i \geq |T| \qquad (11)$$

Condition (11) merely acknowledges that one could perhaps be using a conservative sampling pulse width, much greater than delay through fast paths between some latches.

CONCLUSIONS

It is possible for max-rate pipeline machines to operate at high rates determined by path differences, rather than the conventional maximum delay. The results apply to any technology, but would prove most useful when signal propagation delays approach or exceed circuit delay. In this case velocity of propagating signals need not limit rates if paths are equalized. The approach described would permit increased performance in present systems environments, and would pave the way for entry into the subnanosecond regime where relatively long transmission lines might exist between gate arrays.

REFERENCES

1 M J FLYNN
Very high-speed computing systems
IEEE Proc Vol 54 No 12 December 1966

2 R A ASCHENBRENNER M J FLYNN
G A ROBINSON
Intrinsic multiprocessing
Proc S J C C 1967

3 R A ASCHENBRENNER
Time-phased parallelism
Proc National Electronics Conference 1967

4 G M AMDAHL
Validity of the single processor approach to achieving large scale computing capabilities
Computer Design December 1967 and
Proc S J C C 1967

5 L W COTTEN
Circuit implementation of high-speed pipeline systems
Proc F J C C 1965

6 S F ANDERSON J G EARLE
R E GOLDSCHMIDT D M POWERS
The IBM System/360 Model 91: floating point execution unit
IBM Journal of R & D January 1967

7 D N SENZIG
Observations on high-performance machines
Proc F J C C 1967

8 D L SLOTNICK
Unconventional systems
Computer Design December 1967 and
Proc S J C C 1967

9 W D LEWIS
Microwave logic
Proc of International Symposium on the
Theory of Switching 1957

10 M O PALEY
LSI and the large computer systems designer
International Solid State Circuits Conference 1968

11 A R STRUBE
LSI for high-performance logic applications
International Electron Devices Meeting 1967

12 H H LOOMIS JR
The maximum-rate accumulator
IEEE Transactions on Electronic Computers Vol EC-15
No 4 August 1966

13 J M WIER
Digital data communications techniques
Proc of IRE Vol 49 January-March 1961

14 W R BENNETT J R DAVEY
Data transmission
McGraw-Hill Inc N Y 1965

15 D N ARDEN
Delayed logic and finite state machines
Proc of AIEE Symposium on Switching Theory and
Logical Design September 1961

16 H H LOOMIS JR
Completeness of sets of delayed-logic devices
IEEE Transactions on Electronic Computers Vol EC-14
No 2 April 1965

17 D H CHUNG
The design considerations of picosecond integrated switching circuits
SWIEECO 1966

18 B ELSPAS J GOLDBERG R C MINNICK
R A SHORT H S STONE
Investigation of propagation-limited computer networks
Final Report Air Force Contract AF19(628)-2902
Stanford Research Institute Project 4523 1964

19 J HARTMANIS
Maximal autonomous clocks of sequential machines
IEEE Transactions on Electronics Computers Vol EC-11
No 1 February 1962

Systematic design for modular realization of control functions

by STANLEY M. ALTMAN and ARTHUR W. LO

Princeton University
Princeton, New Jersey

INTRODUCTION

The feasibility and the problems associated with the design of asynchronous digital systems have been variously studied and reported.[2–6] J. B. Dennis has characterized modular design of asynchronous digital systems in the following manner:[1]

The structure of an asynchronous digital system may be divided into the *data flow structure* and the *control structure*. The storage of data, the flow of data and the operations performed on them take place in the data flow structure. The different operations taking place concurrently in the data flow structure are co-ordinated by the control structure.

Each operational unit of the data-flow structure has, in addition to data links (data input lines), a control link connecting it to some part of the control structure. A control link consists of two wires called the *ready line* and the *acknowledge line*. To cause an operator to operate on an input, the input is made available to it (the operator) on the input data link and a ready signal is sent to it (the operator) on the ready line of the control link. After the operation has been performed on the output placed on the output data link the operator returns an acknowledge signal on the acknowledge line. The time difference between the arrival of the ready signal and the return of the acknowledge signal is arbitrary and may depend on the operator, the input, and can even be random so long as the acknowledge signal correctly implies completion of the operation.

The control structure consists of control modules interconnected among themselves and to the data-flow structure through control links. Signals propagate in the forward direction on the ready line. The direction of a control link is the direction in which the signals propagate on its ready line.

In addition to control links a control module can have conditional inputs and outputs. The conditional inputs convey information about the condition of some operation or control modules to this module (and thereby affect the operation of the module), and the conditional outputs convey the condition of this module to other circuits.

Dennis has proposed a set of nine control modules which he has found sufficient to realize all required control functions (these modules are described in detail in Reference 9). While he has found it straightforward to implement control modules without conditional inputs, Dennis has found the design of control modules with conditional inputs quite complex.

In terms of hardware design, the *operation modules* (building blocks of the data-flow structure) have been extensively investigated. The design of registers, adders, transfer devices, and other operational blocks is well established. On the other hand, the basic properties and the logical organization of *asynchronous control modules* (building blocks of the control structure) has not been thoroughly investigated.

This paper introduces properties of a class of asynchronous control modules, whose operation satisfy certain general constraints, and presents a systematic procedure for designing their logical structure. Included in this class are the modules defined by Dennis.

Although the design procedure is derivable from fundamental properties of these modules, it is presented without proof.[9] One result of previous studies is: all but the UNION module are realizable by simple combinational logic (the UNION module requires a single feedback loop).

Asynchronous control modules

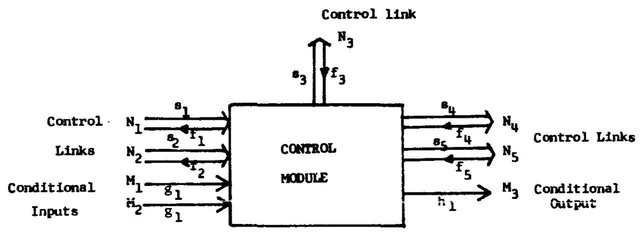
The purpose of this paper is to present a design technique for asynchronous control modules. Therefore, it is necessary that we:

1. Define formally what is meant by asynchronous control module.
2. Define the class of modules studied; i.e., the constraints placed on the control module's operation.
3. Define the model used to represent the control module's operation.

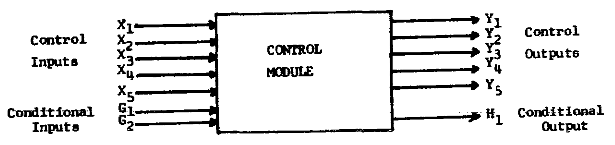
Terminology and notation

The operation of control modules is described in terms of (i) control links and conditional inputs and outputs, and (ii) input and output terminals. Every signal appearing on input and output terminals is binary; i.e., takes on the value of 0 or 1.

A *control link* consists of a pair of wires (s,f); s being the start line and f being the finish line. Control links are classified as either *input* or *output* control links. Input control links have their start lines as input terminals of the module and their finish lines as output terminals of the module. For output control links, the start lines are outputs and the finish lines are inputs (Figure 1 is a block diagram of a control module). A control link is said to be *idle* if its s and f lines carry the same binary signal, and it is said to be *active* if its s and



(a) Control module showing control links and conditional inputs and outputs.



(b) Control module showing input and output terminal states

Figure 1—Generalized block diagrams of control modules
(a) Control module showing control links and conditional inputs and outputs
(b) Generalized block diagrams of control modules

f lines carry different binary signals. The *control link state* (C.L.S.) for a control module with n control links is an ordered n-tuple

$$\mathbf{L} = [L_1 \ldots L_k/L_{k+1} \ldots L_n]$$

where the state of the k input control links appear to the left of the slash and the state of the $(n–k)$ output control links appear to the right of the slash. $L_i = \mathrm{I}$ if s_i and f_i carry the same binary signal and $L_i = \mathrm{A}$ if s_i and f_i carry different binary signals. If every control link is idle, the control link state is defined to be the *quiescent state.*

In terms of the input and output terminals, the *control input state* and the *control output state* are the ordered n-tuples $\mathbf{X} = [X_1 \ldots X_n]$ and $\mathbf{Y} = [Y_1 \ldots Y_n]$ respectively. If N_i is an input control link, then X_i represents the state ($X_i = 0$ or 1) of its start line s_i and Y_i represents the state ($Y_i = 0$ or 1) of its finish line f_i. On the other hand, if N_i is an output control link, then X_i represents the state of f_i and Y_i represents the state of s_i.

In addition to control links, a control module can have conditional inputs and outputs. The *conditional input state,* **G**, and the *conditional output state,* **H**, are the ordered r-tuple $\mathbf{G} = [G_1 \ldots G_r]$ and the ordered k-tuple $\mathbf{H} = [H_1 \ldots H_k]$ respectively. Finally we define the *total input state* and the *total output state* as the vectors, [**X**,**G**] and [**Y**,**H**].

Two major differences exist between control links and conditional inputs and outputs. First, control links are ordered pairs of wires; one is an input terminal of the module and the other an output terminal of the module. No such relationship exists between conditional inputs and outputs. Second, the state of a link is either active or idle, whereas the state of a conditional line is the actual signal it carries.

The control module shown in Figure 1 is used to illustrate the state notation of asynchronous control modules. Table 1 summarizes this notation.

General constraints

There are three general constraints which apply to the operation of each control module unless stated otherwise. These are:

1. Only one input terminal can change at a time.
2. A control link in its idle state permits a change in the state of its s line but not its f line. Similarly, a control link in its active state permits a change in its f line but not its s line.
3. Any change in the conditional input state must

Table I—Terminal state notation of control modules

	Control Links, L_i					*Conditional*		
	Inputs			Outputs		Inputs		Outputs
	N_1	N_2	N_3	N_4	N_5	M_1	M_2	M_3
Input lines, X_i :	s_1	s_2	s_3	f_4	f_5	g_1	g_2	—
Output lines, Y_i :	f_1	f_2	f_3	s_4	s_5	—	—	h_1

Control Link State:	$\mathbf{L} = (L_1L_2L_3/L_4L_5]$	where L_i = I or A
Control Input State:	$\mathbf{X} = [X_1X_2X_3X_4X_5]$	where $X_i = 0$ or 1
Control Output State:	$\mathbf{Y} = [Y_1Y_2Y_3Y_4Y_5]$	where $Y_i = 0$ or 1
Control Input/Output State:	$\mathbf{X/Y} = [X_1X_2X_3X_4X_5/Y_1Y_2Y_3Y_4Y_5]$	
Conditional Input/Output State:	$\mathbf{G/H} = [G_1G_2/H_1]$	where $G_i = 0$ or 1; $H_i = 0$ or 1
Total Input State:	$[X_1X_2X_3X_4X_5,\ G_1G_2]$	
Total Output State:	$[Y_1Y_2Y_3Y_4Y_5,\ H_1]$	

be completed before the *C.L.S.* is allowed to change.

Since the control modules are asynchronous circuits the design of logic circuits which perform the desired modes of operation can be carried out by the classical primitive flow table approach. This approach, however, proves to be highly inefficient. For example the *SELECT* module (to be discussed below) is described by a primitive flow table with 32 columns and at least 16 rows. A more efficient procedure is to construct an *action graph* of the control module.

Action graph

The design of a specific control module is derived from the *action graph,* which is strongly connected, representing the step-by-step operation of the module as described by its word statement. Each node of the graph represents a *control link state,* and each directed edge of the graph represents a specific operation which transforms one control link state into another.

Nodes representing control link states in which the conditional input state changes are called *transfer nodes.* These are the nodes at which decisions are made among alternative modes of operation.

Every directed edge of the graph has associated with it a pair of functions (Δ,θ) where Δ transforms one C.L.S. into another C.L.S. and θ specifies the affect of changes in the conditional input state on the operation of the modules. This ordered pair is called the *edge operator.*

The arguments of Δ specify which control input terminal and which control output terminals undergo a state change during the transition specified by the edge. $\Delta(x/y)$ operates on both the control input state and the control output state and therefore can be decomposed into the input control operator, δx, and the output control operator, δy; i.e., $\Delta(x/y) = \delta x/\delta y$. For example, the control operator $\Delta(x_a/y_by_c) = \delta x_a/\delta y_by_c$ transforms the control input/output state $[X_aX_bX_c/Y_aY_bY_c]$ into the control input/output state $[\bar{X}_aX_bX_c/Y_a\bar{Y}_b\bar{Y}_c]$ where $\bar{X}_i$ and $\bar{Y}_i$ are respectively complements of X_i and Y_i.

If an edge originates at a tranfser node, then $\theta = G_i$ where G_i is the conditional input state *before* the C.L.S. is permitted to change. For an edge originating from a node where the conditional inputs cannot change, θ is replaced by a dash (—). If the control module does not have conditional inputs, then the edge operators (Δ,θ) reduce to Δ.

Whenever a control module has conditional outputs, every node of the action graph has a unique value of the conditional output state, H, associated with it. The method of design is best described by an example.

Design of the SELECT module

The block diagram of the *SELECT* module, which has four control links, one conditional input, and no conditional outputs, is shown in Figure 2. The function of this module is to initiate one of two alternative modes of operation (represented by activating either N_3 or

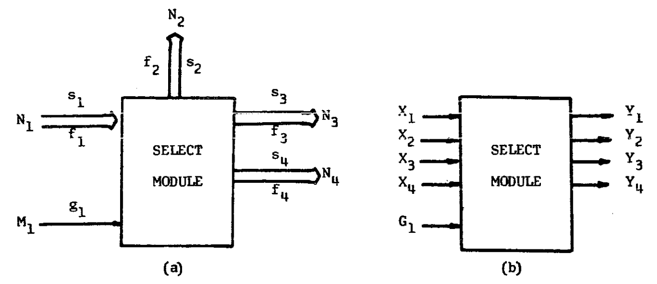


Figure 2—Block diagram of the SELECT module

N_4). The operation of the *SELECT* module follows the following word statement

Word statement of the *SELECT* module

1. On receiving a start signal on control link N_1 a start signal is immediately sent on control link N_2.
2. When a finish signal is eventually received on N_2 a start signal is sent on N_3 or N_4 according as the value of the conditional input state, **G**, is [1] or [0] respectively. N_3 and N_4 cannot both be active at the same time.
3. When a finish signal is received from N_3 or N_4, a finish signal is transmitted on link N_1. The module has now returned to its quiescent state, and a fresh start signal may now be received on N_1.

Design procedure

The design of the *SELECT* module begins with the construction of an action graph. The control module is said to be in its *Quiescent State* when all its control links are idle. The module remains in its *Q.S.* until the input line s_1 changes its state (Word Statement 1). Thus the first step in constructing the action graph is to draw the quiescent-state node [I/III] as shown in Figure 3a. A change in s_1 causes a change on s_2 (Word Statement 1). This is represented by the directed edge labeled $(\Delta(s_1/s_2)$,-) as shown in Figure 3b, where the operation of the module is independent of the conditional input state. The *edge operator* $(\Delta(s_1/s_2)$,-) transfers the control link state of the module from the node [I/III] to the node [A/AII] in the action graph. A change on f_2 causes a change on s_3 if the conditional input state is [1] or a change on s_4 if the conditional input state is [0] (Word Statement 2). This is represented by the edge operator $(\Delta(f_2/s_3), 1)$ which transfers the node [A/AII] to the node [A/IAI] and the edge operator $(\Delta(f_2/s_4), 0)$ which transfers the node [A/AII] to the node [A/IIA] as shown in Figure 3c. The node [A/AII]

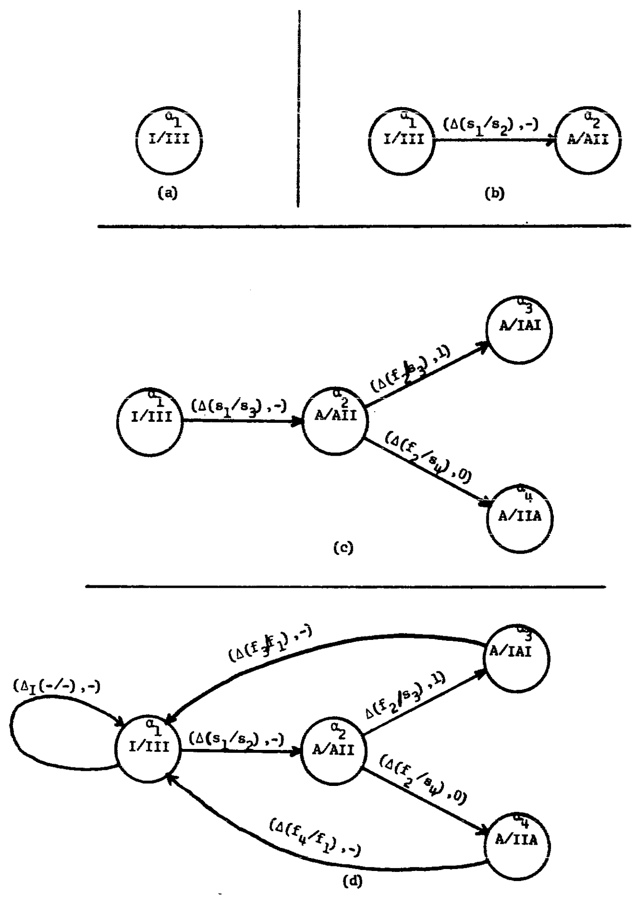


Figure 3—Determination of SELECT module's action graph

is therefore a transfer node. If the conditional input state [1], then a change on f_3 causes a change on f_1 (Word Statement 3). This is represented by the edge operator $[\Delta(f_3/f_1)$, —) which transfers the node [A/IAI] to the node [I/III]. If the conditional input state were [0], then a change on f_4 causes a change on f_1 (Word Statement 3). This is represented by the edge operator $(\Delta(f_4/f_1)$, —) which transfers the node [A/IIA] to the node [I/III]. Both cases are summarized in Figure 3d. The action graph is now complete. A self-loop with the edge operator $(\Delta_I$(—/—), —) is drawn on the quiescent-state node. This artifice is introduced for the convenience of later discussion. The operator $(\Delta_I$(—/—), —), representing no change of input or output, is defined as the *identity operator*.

Simplified action graph

Because of the relationship that exists between the control input state, control output state, and the control link state, any two of the variables uniquely determines the third variable. From the definition of the Control Link State,

$$Y_i = X_i, \text{ if } L_i = \text{I} ;$$

$$Y_i = \bar{X}_i, \text{ if } L_i = \text{A} .$$

As a result, the action graph contains redundant information. This graph can be simplified by replacing every control operator $\Delta(x/y)$ by its input component δx without any loss of information. Figure 4 is the simplified action graph of the *SELECT* module.

The action graph indicates that the operation of the module can be described by a sequence of input operators. We define the *directed path operator* to be the product of the edge operators that correspond to the directed edges traversed in going from one node of the graph to another node of the graph. A path which begins and ends on the same node is called a *cycle* and a path operator corresponding to a cycle is called a *cycle operator*. Figure 5 illustrates the different cases that can arise.

The conditional operator $\theta = \mathbf{G}_i \ldots \mathbf{G}_j\mathbf{G}_k$ can be simplified and replaced by $\theta = \mathbf{G}_k$. This simplification, called the reduced form of θ, is possible because θ transforms any conditional input state into the conditional input state $\mathbf{G}_k$ determined by the last transfer node encountered, independent of $\mathbf{G}_i, \ldots, \mathbf{G}_j$.

The control portion of the input path operator $(\delta_a\delta_b\delta_c\delta_d, \theta)$ can be represented by δ_{abcd}, where the string of symbols abcd is said to be the argument of δ. The argument of every path operator can be expressed in terms of a minimal product of the input variables. The minimal product has the form $x_1^{n1}x_2^{n2}x_4^{n3} \ldots$, where n_i $i = 1,2,3, \ldots$ is the number of times n_i appears in the argument of δx. If n_i is even, then $x_i^{n_i}$ produces the same change as (—), i.e., no change at all. If n_i is odd, then $x_i^{n_i}$ produces the same change as x_i.

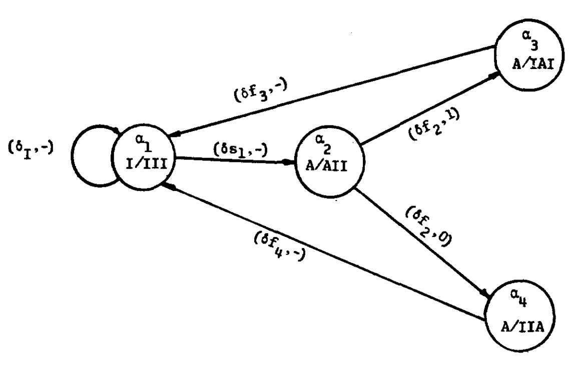


Figure 4—Simplified action graph of the SELECT module

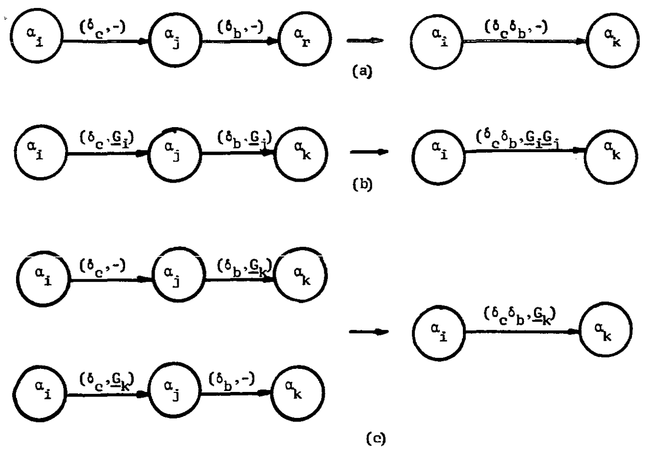


Figure 5—Generalized properties of the input edge operator

An input path operator is said to be in its reduced form if every control input variable in the argument of δ appears once or not at all. The canonical reduced form of $\delta x_1^{n1}x_2^{n2}x_3^{n3} \ldots$ is $\delta x_1^{\beta 1}x_2^{\beta 2}x_3^{\beta 3} \ldots$ where $\beta_i = 1$ if x_i is present and $\beta_i = 0$ if x_i is not present. For convenience the reduced form of $\delta x_1^{\beta 1}x_2^{\beta 2} \ldots x_m^{\beta m}$ is identified as δ_k where k is chosen to be the *octal* equivalent of the binary number $\beta_1\beta_2 \ldots \beta_m$. δ_0 is the identity operator δ_I. If a control module has n control links, then there exists 2^n distinct reduced-form operators. The set of 2^n reduced-form control operators is denoted by $\mathcal{R}$. Since the *SELECT* module has four control links, $\mathcal{R}$ contains

$\delta_{00} = \delta_I$	$\delta_{10} = \delta x_1$
$\delta_{01} = \delta x_4$	$\delta_{11} = \delta x_1 x_4$
$\delta_{02} = \delta x_3$	$\delta_{12} = \delta x_1 x_3$
$\delta_{03} = \delta x_3 x_4$	$\delta_{13} = \delta x_1 x_3 x_4$
$\delta_{04} = \delta x_2$	$\delta_{14} = \delta x_1 x_2$
$\delta_{05} = \delta x_2 x_4$	$\delta_{15} = \delta x_1 x_2 x_4$
$\delta_{06} = \delta x_2 x_3$	$\delta_{16} = \delta x_1 x_2 x_3$
$\delta_{07} = \delta x_2 x_3 x_4$	$\delta_{17} = \delta x_1 x_2 x_3 x_4$

At this point several properties of asynchronous control modules can be introduced. These results are stated without proof. Unless specifically stated to the contrary, all operators are assumed to be in reduced form.

The action graph must be strongly connected because the module returns to a starting state when its required operation is completed. In general, this state is the quiescent state. Every node in the action graph has associated with it a maximal set of distinct input cycle operators. For any node α_k, denote its corresponding set of cycle operators by $\mathfrak{D}_k$. Also associated with α_k is the set $\mathcal{G}_k$, where $\mathcal{G}_k$ is the set of states that the conditional inputs can assume when the control module is in the *C.L.S.* rerresented by α_k; e.g., $\mathcal{G}_1$ for the *SELECT* module is $\{0,1\}$.

The action graph of the *SELECT* module is shown in Figure 4. Beginning at node α_1 (quiescent node) there exists a cycle which passes through nodes α_2 and α_3 exactly once. Associated with this cycle is the operator $(\delta_{16},1)$. Call this cycle p_1. The cycle which begins at α_1 and passes through nodes α_2 and α_4 exactly once, has associated with it the cycle operator $(\delta_{15}, 0)$. Call this cycle p_2. $(\delta_{16},1)$ and $(\delta_{15},0)$ belong to $\mathfrak{D}_1$ by definition. Consider the paths that are formed by the juxtaposition of p_1 and p_2; in particular $p_1p_2p_2$ and $p_2p_1p_1$. Associated with these paths are the operators $(\delta_{16},0)$ and $(\delta_{15},1)$ respectively. These operators also belong to $\mathfrak{D}_1$. It can be shown that $\mathfrak{D}_1$ contains

$$\{(\delta_0,0), (\delta_0,1), (\delta_3,0), (\delta_3,1), (\delta_{15},0), (\delta_{15},1), (\delta_{16},0), (\delta_{16},1)\}$$

This example illustrates the following Theorem.

Theorem 1. If the operator $(\delta_i, \mathbf{G}_j) \epsilon \mathfrak{D}_k$, then the operators $(\delta_i, \mathbf{G}_m) \epsilon \mathfrak{D}_k$, for all like $\mathbf{G}_m \epsilon \mathcal{G}_k$.

From Theorem 1 set $\mathfrak{D}_k$ can be partitioned into blocks, such that each block is identified by a distinct control cycle operator. Let $\mathfrak{D}$ be the set of distinct control cycle operators that appear in $\mathfrak{D}_k$. A second partition of $\mathfrak{D}_k$ can be formed by grouping together all of the cycle operators whose conditional input state are the same; i.e., partition $\mathfrak{D}_k$ into blocks $\mathfrak{D}_k(\mathbf{m}) = \{\mathfrak{D},\mathbf{G}_m\}$ for all $\mathbf{G}_m \epsilon \mathcal{G}_k$. $\mathfrak{D}$ is the same for all nodes of the action graph. In terms of $\mathfrak{D}$ and $\mathcal{G}_k$, $\mathfrak{D}_k$ is represented by $\mathfrak{D}_k = \{\mathfrak{D},\mathcal{G}_k\}$. If the control module does not have any conditional inputs, then the maximal set of distinct cycle operators for every node of the graph is $\mathfrak{D}$.

Consider a control module with n control links and r conditional input states, $\mathbf{G}_1, \ldots, \mathbf{G}_r$. $\mathcal{R}_G$ denote the set of $r(2^n)$ distinct reduced form input path operators $\mathcal{R}(\mathbf{1}), \mathcal{R}(\mathbf{2}), \ldots, \mathcal{R}(\mathbf{r})$ where $\mathcal{R}(\mathbf{i}) = \{R,\mathbf{G}_i\}$ for $i = 1, \ldots, r$,

Theorem 2. The simplified action graph partitions $\mathcal{R}_G$ into the kr blocks of input path operators $\{D_j(\mathbf{i})\}$

$$i = 1, \ldots, r$$
$$j = 1, \ldots, k$$

where

$$\begin{aligned}
D_1(\mathbf{i}) &= \mathfrak{D}(\mathbf{i}) \\
D_2(\mathbf{i}) &= (\delta_a,\mathbf{G}_i)\,\mathfrak{D}(\mathbf{i}) && (\delta_a,\mathbf{G}_i) \notin D_1(\mathbf{i}) \\
&\vdots && \vdots \\
D_k(\mathbf{i}) &= (\delta_c,\mathbf{G}_i)\,\mathfrak{D}(\mathbf{i}) && (\delta_c,\mathbf{G}_i) \notin \{\mathfrak{D}_1(\mathbf{i}) + \ldots + D_{k-1}(\mathbf{i})\}
\end{aligned}$$

for $\mathbf{i} = 1, \ldots, r$.

The sets $D_j(\mathbf{i})$, $\mathcal{R}(\mathbf{i})$ and $\mathcal{R}_G$ satisfy the following relationship

$$\mathcal{R}(\mathbf{i}) = \{D_1(\mathbf{i}), D_2(\mathbf{i}), \ldots, D_k(\mathbf{i})\} \qquad i = 1, \ldots, r$$

and

$$\mathcal{R}_G = \{\mathcal{R}(\mathbf{1}), \mathcal{R}(\mathbf{2}), \ldots, \mathcal{R}(\mathbf{r})\}$$

If each block, $D_j(\mathbf{i})$, contains m distinct input path operators, then k, and m satisfy the relationship $km = 2^n$, $k = 2^{n_1}$, $m = 2^{n_2}$, and n_1 and n_2 are positive integers such that $n_1 + n_2 = n$, for $\mathbf{i} = 1, \ldots, r$.

The set $\mathfrak{D}$ can be determined using signal flow graph theory[7,8] (see Appendix). Since $\mathfrak{D}$ is independent of $\mathbf{G}$ only the control input operator portion δ_i, of the input path operator need be shown on the simplified action graph. Figure 6 shows the determination of $\mathfrak{D}$ for the *SELECT* module. From Figure 6 $\mathfrak{D}$ was found to contain

$$\mathfrak{D} = \{\delta_{00}, \delta_{03}, \delta_{15}, \delta_{16}\}$$

Corollary 1: The input edge operator (δ_p,θ), transforms the block of input path operators $D_i(\mathbf{k}) = \{D_i,\mathbf{G}_k\}$ into the block of input path operator $D_j(\mathbf{r}) = \{D_j,\mathbf{G}_r\}$.

Case 1: The node α is not a transfer node, $\theta = (-)$

$$\begin{aligned}
[D_i(\mathbf{k})]\,(\delta_p,-) &= \{D_i,\mathbf{G}_k\}\,(\delta_p,-) \\
&= \{D_i\delta_p, \mathbf{G}_k\} \\
&= \{D_j, \mathbf{G}_k\} \\
&= D_j(\mathbf{k})
\end{aligned}$$

where D_j is the reduced form of $D_i\delta_p$.

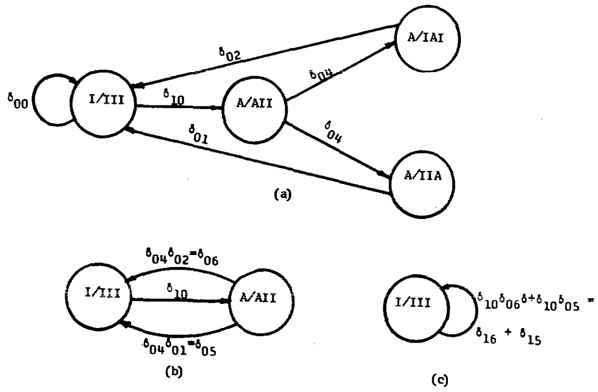


Figure 6—Determination of $\mathfrak{D}$ for the SELECT module

Case 2: The node α is a transfer node, $\theta = \mathbf{G}_r$

$$\begin{aligned}[\mathrm{D}_i(\mathbf{k})]\,(\delta_p,\mathrm{G}_r) &= \{\mathrm{D}_i,\mathrm{G}_k\}\,(\delta_p,\mathrm{G}_r)\\ &= \{\mathrm{D}_i\delta_p,\mathrm{G}_k\mathrm{G}_r\}\\ &= \{\mathrm{D}_j,\mathrm{G}_r\}\\ &= \mathrm{D}_j(\mathbf{r})\end{aligned}$$

Theorem 2 states that the action graph partitions the set of all possible control operators into k blocks, each block containing the same number of elements. One of these blocks is $\mathfrak{D}$. If $\mathfrak{D}_k$ is known for any node α_k, then the maximal set of distinct path operators associated with the directed paths from node α_k to every other node in the graph can be computed as shown in Corollary 1. The assignment of blocks from the partition of $\mathfrak{R}_G$ to the edges of the graph provide the basis for (i) deciding if the module is realizable by combinational logic, and (ii) constructing the *Karnaugh Maps* used to design the module's logical structure.

Theorem 3: A control module is realizable by a combinational circuit if and only if every block, $\mathrm{D}_i(\mathbf{k})$, appears at most once in the action graph, with multiple appearance occurring only on edges incident on the same node. An action graph which satisfies these conditions is said to be a Type I action graph.

The design of the *SELECT* module can now be completed. Using $\mathfrak{D}$ to compute the partition on $\mathfrak{R}$, we find

$$\mathrm{D}_1 = \delta_{00}\mathfrak{D} = \{\delta_{00}, \delta_{03}, \delta_{15}, \delta_{16}\}$$

$$\mathrm{D}_2 = \delta_{01}\mathfrak{D} = \{\delta_{01}, \delta_{02}, \delta_{14}, \delta_{17}\} \text{ where } \delta_{01} \notin \mathrm{D}_1$$

$$\mathrm{D}_3 = \delta_{04}\mathfrak{D} = \{\delta_{04}, \delta_{07}, \delta_{11}, \delta_{12}\} \text{ where } \delta_{04} \notin \{\mathrm{D}_1 + \mathrm{D}_2\}$$

$$\mathrm{D}_4 = \delta_{05}\mathfrak{D} = \{\delta_{05}, \delta_{06}, \delta_{10}, \delta_{13}\} \text{ where } \delta_{05} \notin \{\mathrm{D}_1 + \mathrm{D}_2 + \mathrm{D}_3\}$$

where the operator "+" is the union or sum operator.

The action graph of the *SELECT* module partitions $\mathfrak{R}_G$ into the eight blocks

$$\mathrm{D}_1(\mathbf{0}) = \{\mathrm{D}_1, \mathbf{0}\} \qquad \mathrm{D}_1(\mathbf{1}) = \{\mathrm{D}_1, \mathbf{1}\}$$

$$\mathrm{D}_2(\mathbf{0}) = \{\mathrm{D}_2, \mathbf{0}\} \qquad \mathrm{D}_2(\mathbf{1}) = \{\mathrm{D}_2, \mathbf{1}\}$$

$$\mathrm{D}_3(\mathbf{0}) = \{\mathrm{D}_3, \mathbf{0}\} \qquad \mathrm{D}_3(\mathbf{1}) = \{\mathrm{D}_3, \mathbf{1}\}$$

$$\mathrm{D}_4(\mathbf{0}) = \{\mathrm{D}_4, \mathbf{0}\} \qquad \mathrm{D}_4(\mathbf{1}) = \{\mathrm{D}_4, \mathbf{1}\}$$

Assigning the set of input path operators, $\{\mathrm{D}_1(\mathbf{0}), \mathrm{D}_1(\mathbf{1})\}$, to the self-loop of the quiescent node, results in the following blocks of input path operators being assigned to the edges of the action graph

Edge Connecting Node to Node	Block of Input Path Operator Assigned to Connecting Edge
$\alpha_1 \rightarrow \alpha_2$:	$(\delta_{10},—)\,\{\mathrm{D}_1(\mathbf{0}),\mathrm{D}_1(\mathbf{1})\} = \{\mathrm{D}_4(\mathbf{0}),\mathrm{D}_4(\mathbf{1})\}$
$\alpha_2 \rightarrow \alpha_3$:	$(\delta_{04},\ \mathbf{1})\,\{\mathrm{D}_4(\mathbf{0}),\mathrm{D}_4(\mathbf{1})\} = \{\mathrm{D}_2(\mathbf{1})\}$
$\alpha_2 \rightarrow \alpha_4$:	$(\delta_{04},\ \mathbf{0})\,\{\mathrm{D}_4(\mathbf{0}),\mathrm{D}_4(\mathbf{1})\} = \{\mathrm{D}_2(\mathbf{0})\}$
$\alpha_3 \rightarrow \alpha_1$:	$(\delta_{02},—)\,\{\mathrm{D}_2(\mathbf{1})\} = \{\mathrm{D}_1(\mathbf{1})\}$
$\alpha_4 \rightarrow \alpha_1$:	$(\delta_{01},—)\,\{\mathrm{D}_2(\mathbf{0})\} = \{\mathrm{D}_1(\mathbf{0})\}$

This assignment is summarized in Figure 7.

From Figure 7 the *SELECT* module is found to satisfy Theorem 3 and is therefore realizable by a combinational circuit. It should be noted that up to this point the description of the operation of the control module and the determination that it is realizable by a combinational circuit has been accomplished independent of the binary (0 or 1) values that appear at the control module's inputs and outputs. However, the logical design of the control module in terms of a complete set of logical primitives connectives must refer to the total input/output states. For our design we shall assume that one of the quiescent input state is [00,0].

Using the total input state [00,0] the block of total input states and the corresponding block of control output states associated with each node in the action graph are computed and summarized in Table II.

Table II—The control link states and the total input/output states of the *SELECT* module

TOTAL INPUT STATES	PRESENT CONTROL LINK STATES	TOTAL OUTPUT STATES
$\mathrm{J}_1(\mathbf{0})$:{(00,03,15,16);0}	α_1:[I/III]	Z_1:{00,03,15,16}
$\mathrm{J}_1(\mathbf{1})$:{(00,03,15,16);1}	α_1:[I/III]	Z_2:{00,03,15,16}
$\mathrm{J}_2(\mathbf{0})$:{(01,02,14,17);0}	α_4:[A/IIA]	Z_3:{10,13,05,06}
$\mathrm{J}_2(\mathbf{1})$:{(01,02,14,17);1}	α_3:[A/IAI]	Z_4:{13,10,06,05}
$\mathrm{J}_4(\mathbf{0})$:{(05,06,10,13);0}	α_2:[A/AII]	Z_5:{11,12,04,07}
$\mathrm{J}_4(\mathbf{1})$:{(05,06,10,13);1}	α_2:[A/AII]	Z_6:{11,12,04,07}

$$\mathrm{J}_i(\mathrm{j}) = \mathrm{D}_i(\mathrm{j})\ [00,0]$$

Since $\mathrm{D}_3(\mathbf{0})$ and $\mathrm{D}_3(\mathbf{1})$ do not appear in the action graph the cells in the Karnaugh Maps identified by

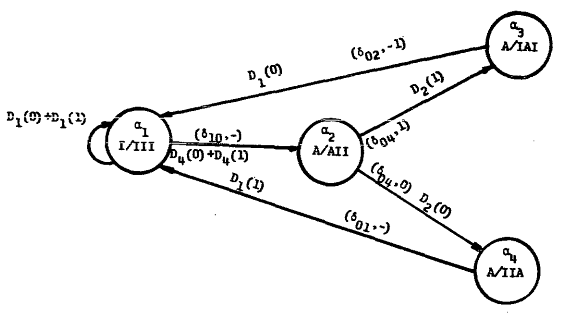


Figure 7—Assignment of input path operator blocks for the SELECT module

(a) f_1

$g_1=0$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	1	0	1
01	d	1	d	1
11	0	1	0	1
10	0	d	0	d

$g_1=1$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	1	0	1
01	d	1	0	1
11	0	1	0	1
10	0	d	0	1

(b) s_2

$g_1=0$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	0	0	0
01	d	0	d	0
11	1	1	1	1
10	1	d	1	d

$g_1=1$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	0	0	0
01	d	0	d	0
11	1	1	1	1
10	1	d	1	d

(c) s_3

$g_1=0$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	0	1	1
01	d	0	d	1
11	0	0	1	1
10	0	d	1	d

$g_1=1$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	1	1	0
01	d	0	d	1
11	1	0	0	1
10	0	d	1	d

(d) s_4

$g_1=0$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	0	1	1
01	d	1	d	0
11	1	1	0	0
10	0	d	1	d

$g_1=0$

s_1f_2 \ f_3f_4	00	01	11	10
00	0	1	1	0
01	d	1	d	0
11	0	1	1	0
10	0	d	1	d

Figure 8—Karnaugh maps of SELECT module

$J_3(0)$ and $J_3(1)$ have "don't care" (d) entries (see Figure 8). It should be noted that the octal equivalent of the control input states belonging to $J_i(j)$ are the octal subscripts, k, of all the $\delta_k \in D_i(j)$.

The Boolean functions relating the output variables f_1, s_2, s_3, and s_4 to the input variables s_1, f_2, f_3, f_4, and g_1 (expressed by the Karnaugh Maps in Figure 8) satisfy the word statement that describes the *SELECT* module's modes of operation. These Karnaugh Maps can now be used to construct a hazard-free combinational circuit realization of the *SELECT* module which satisfies given cost, fan-in, and fan-out constraints.

CONCLUDING REMARKS

As indicated in Theorem 3 the action graph of a control module realizable by a combinational circuit is classified as Type I and it has the property that every node has a unique set of operators associated with it (block of operators assigned to the edges which terminate on the node). In terms of the action graph there are two cases in which a control module requires internal memory to perform its specified function.

Case 1: Every edge that has the same block of distinct operators assigned to them are incident upon distinct nodes. Such an action graph is classified as Type II.

Case 2: More than one edge originating from node α_i has the same block of operators assigned to it, and the nodes they terminate on are distinct. Such an action graph is classified as Type III.

Although both Type II and Type III action graphs require internal memory, they can be designed directly from the action graph by (i) converting Type III graphs into Type II graphs, and (ii) converting Type II graphs into Type I graphs. The algorithm for performing the conversions are straightforward, but are too lengthy to be presented here. The algorithms are described in detail in Reference 9.

ACKNOWLEDGMENT

The authors would like to express their appreciation to Dr. John Bruno for many helpful comments. This work was partially supported by the National Science Foundation.

REFERENCES

1 J B DENNIS
Computation structure (lecture notes)
COSINE Committee Lecture Series July 1968

2 D E MULLER W S BARTKY
A theory of asynchronous circuits
The Annals of the Computation Laboratory of Harvard University 1959

3 G ESTRIN
Organization of computer systems—fixed plus variable structure computer
Proc W J C C 1960

4 G ESTRIN B RUSSELL R TURN J BIBB
Parallel processing in a restructurable computer system
IEEE Transactions of Electronic Computers 1963

5 W A CLARK
Macromodular computer systems
Proc S J C C 1977
6 S M ORNSTEIN M STUKI W A CLARK
A function description of macromodules
Proc S J C C 1967
7 S J MASON
Feedback theory—some properties of signal flow graphs
Proc IRE 1953
8 J A BRZOZOWSKI E J McCLUSKEY
Signal flow graph techniques for sequential circuit state diagrams
IEEE Trans on Electronic Computers 1963
9 S M ALTMAN A W LO
The properties and design of asynchronous control modules
Princeton University's Computer Science Laboratory
TR No 71 1968

APPENDIX

Determination of 𝔇 from the simplified action graph

𝔇 can be determined directly from the action graph through the use of signal flow graph theory.[7,8] If all the nodes but one are removed from the action graph, then the path operators appearing on the self-loop of the remaining node, when starred, yields 𝔇. The essential operations performed on the graph are summarized in Figure 9.

The "*" operator shown in Figure 9 has the following properties:

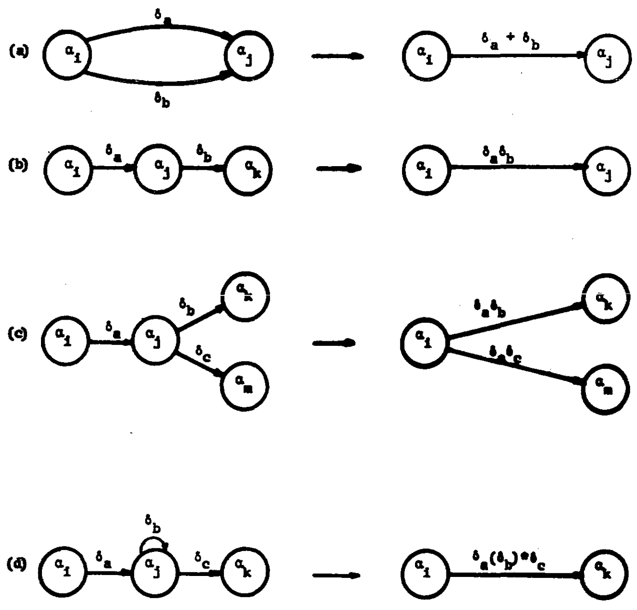


Figure 9—Signal flow graph reduction rules

1. $\delta_a^* = \delta_o + (\delta_a) + (\delta_a)^2 + (\delta_a)^3 + \dots$
   $= \delta_o + \delta_a$

2. $(\delta_a + \delta_b)^* = \delta_a + (\delta_a + \delta_b) + (\delta_a + \delta_b)^2$
   $= \delta_o + \delta_a + \delta_b + \delta_a\delta_b + (\delta_a\delta_b)^2 + \dots$
   $= \delta_o + \delta_a + \delta_b + \delta_a\delta_b$

Optimizing floating point arithmetic via post addition shift probabilities

by JAMES A. FIELD

University of Waterloo
Waterloo, Ontario, Canada

INTRODUCTION

In many computers floating point arithmetic operations are performed by subprograms: software packages in the case of most small computers, and micro-programmed read-only memories in some larger systems. In such a subprogram there are normally several free choices as to which set of conditions gets a speed advantage. If this advantage is given to the most probable case then there will be an increase in system performance with no increase in cost.

One area in which this type of optimization is possible is in the processing of binary floating point addition and subtraction. Here there exist two possible shift operations, first to align the binary points before addition or subtraction, and second, to normalize the result. In processing these shifts there are several options as to method, and sequencing of operations within a given method. To choose the variation that optimizes the program it is necessary to know the probability of occurrence of the various length shifts possible.

Sweeney[1] has reported experimentally determined distributions for shift lengths in alignment and normalization. Unfortunately the data for normalization was presented as total values. Subprogram optimization requires normalization shift length probabilities given that a specific alignment shift occurred.

This paper presents a method for estimating the required probabilities, and an example of their application in subprogram optimization.

Shift length probabilities

A common representation of the fractional part of a floating point number is a sign bit plus a normalized n bit true magnitude. This form will be used in the analysis. The form of the exponent is not of concern.

In normalized numbers the leading bit is always a one. It will be assumed that in all other bits there is equal probability of a one or zero. Appendix A gives the reasons for this assumption.

For purpose of analysis the addition operation can be divided into five cases. These are considered in the following sections. While only the addition operation will be specifically considered, the results are also applicable to subtraction as it is just addition with the sign bit complemented.

The following representations for shift length probabilities will be used:

P_{-1}—the probability that a one bit right shift is required for normalization.
P_0 —the probability that no normalization shift is required.
P_i —the probability that an i bit left shift is required for normalization $(i > 0)$.

Like signs, equal exponents

When the numbers have equal exponents they may be added immediately since no alignment shift is required. With the leading bit of both words being a one, the sum will always contain $(n + 1)$ bits. Thus a one bit right shift will always be required to normalize the result of the addition. Therefore

$$P_{-1} = 1 \qquad (1)$$

Like signs, unequal exponents

When the exponents differ the smaller number must be shifted right until the binary points are aligned. Figure 1 shows the situation after the alignment shift of $s = (n - m)$ bits has taken place. The x's indicate bits that may with equal probability be either one or zero. A $(n + 1)$ bit result will occur, requiring a one bit

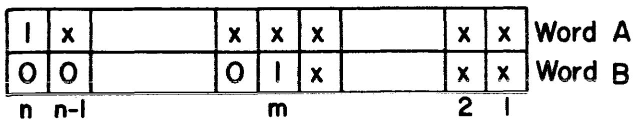


Figure 1—Numbers following binary point alignment

right shift for normalization, if and only if there is a carry into bit n.

Defining

A_j = j^{th} bit of word A

B_j = j^{th} bit of word B (after binary point alignment)

C_j = carry into j^{th} position

S_j = j^{th} bit of unnormalized sum

Then

$$\Pr(C_{j+1} = 1) = \tfrac{1}{4}[1 - \Pr(C_j = 1)] + \tfrac{3}{4}\Pr(C_j = 1)$$

$$= \tfrac{1}{4} + \tfrac{1}{2}\Pr(C_j = 1)$$

$$= \frac{1}{2} - \frac{1}{2^{j+1}} + \frac{\Pr(C_1 = 1)}{2^j}\,;$$

$$m - 1 \geq j \geq 1 \quad (2)$$

and, since $B_m = 1$,

$$\Pr(C_{m+1} = 1) = \tfrac{1}{2}[1 - \Pr(C_m = 1)] + \Pr(C_m = 1)$$

$$= \tfrac{1}{2} + \tfrac{1}{2}\Pr(C_m = 1)$$

If the alignment shift was one bit then C_{m+1} is C_n. However, for alignment shifts greater than one bit, only if all bits A_{n-1} through A_{m+1} are ones will a carry propagate from the $(m + 1)^{th}$ to the n^{th} bit.

Hence:

$$\Pr(C_n = 1) = \Pr(C_{m+1} = 1) \qquad ; m = n - 1$$

$$= \Pr(C_{m+1} = 1) \prod_{j=m+1}^{n-1} \Pr(A_j = 1)$$

$$; m < n - 1$$

$$= \frac{1}{2^{n-m-1}}\Pr(C_{m+1} = 1)$$

$$; m \leq n - 1$$

Now, with

$$\Pr(C_1 = 1) = R/2$$

where

R = 1 for systems where word B is rounded after alignment, and

R = 0 for systems where word B is truncated after alignment it

follows that

$$P_{-1} = \Pr(S_{n+1} = 1) = \Pr(C_n = 1) = \frac{3}{4} \cdot \frac{1}{2^{s-1}}$$

$$+ \frac{R - 1}{2^n} \quad ; n > s \geq 1 \quad (3a)$$

If there is rounding, overflow can occur for an alignment shift of n bits if word A is all ones, hence

$$P_{-1} = \Pr(S_{n+1} = 1) = \frac{R}{2^{n-1}} \qquad ; s = n \qquad (3b)$$

If the exponents differ by n (n + 1 when rounding) or more then no shifting is required since the larger number is the result. Hence

$$P_{-1} = 0 \qquad ; s > n \qquad (3c)$$

In all cases the only alternative to a one bit right shift is no shift, therefore

$$P_0 = 1 - P_{-1} \qquad ; s \geq 1 \qquad (3d)$$

Unlike signs, equal exponents

In Figure 2 is shown a tabulation of all possible combinations of two n bit words with unlike signs. If all bits but the most significant may be one or zero with equal probability it follows that all the combinations listed in Figure 2 are equally probable. Thus to obtain the probability of having exactly i leading zeros after forming the sum requires only that the number of such sums be counted.

When the sum is zero no shift is required, while a sum with i leading zeros requires an i bit left shift for normalization. Hence

$$P_0 = \Pr(\text{zero result}) = \frac{1}{2^{n-1}} \qquad (4a)$$

```
 1111    1111    1111  ....  1111    1111
-1111   -1110   -1101       -1001   -1000
 0000    0001    0010        0110    0111

 1110    1110    1110  ....  1110    1110
-1111   -1110   -1101       -1001   -1000
-0001    0000    0001        0101    0110

  .       .       .           .       .

  .       .       .           .       .

 1001    1001    1001  ....  1001    1001
-1111   -1110   -1101       -1001   -1000
-0110   -0101   -0100        0000    0001

 1000    1000    1000  ....  1000    1000
-1111   -1110   -1101       -1001   -1000
-0111   -0110   -0101       -0001    0000
```

Figure 2—Array of all possible combinations of two n-bit normalized numbers with unlike signs and equal exponents (n = 4)

and for non-zero results

$$P_i = \text{Pr(exactly i leading zeros)} = \frac{2}{2^{2(n-1)}} \sum_{j=2^{n-i-1}}^{2^{n-i}-1} (2^{n-1} - j)$$

$$= \frac{1}{2^{i-1}} \left(1 - \frac{3}{2^{i+1}} + \frac{1}{2^n}\right) \quad ; \quad 1 \leq i < n \qquad \text{(4b)}$$

Unlike signs, exponents differ by one

This case requires that the smaller number, after the alignment shift, be subtracted from the larger number. This subtraction may be considered as the addition of the one's complement plus one in the least significant position. The bit alignments are shown in Figure 3. It can be seen that there will be at least one leading zero if $C_n = 0$.

Considering the extra one added into the least significant position as a carry yields

$$\Pr(C_1 = 1) = 1 - R/2$$

where R is defined as before, and since Equation 2 applies,

$$\Pr(C_{j+1} = 1) = \frac{1}{2} + \frac{1 - R}{2^{j+1}} \; ; \; m - 1 \geq j \geq 1$$

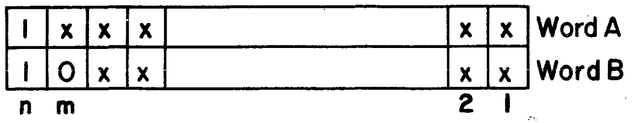


Figure 3—Numbers following binary point alignment and one's complementing of word B (exponents differ by one)

and, since $B_m = 0$,

$$\Pr(C_{m+1} = 1) = \frac{1}{2} \Pr(C_m = 1)$$

$$= \frac{1}{4} + \frac{1 - R}{2^{m+1}} \qquad (5)$$

with

$$\Pr(S_n = 1) = \Pr(C_{m+1} = 1)$$

Hence

$$\Pr(S_n = 0) = \frac{3}{4} - \frac{1 - R}{2^n}$$

For the first i bits of the sum to be zero requires that C_{n-i+1} and A_m be zero, and for i greater than two, that A_{m-1}, B_{m-1}, . . ., A_{n-i+1}, B_{n-i+1} also be zero. Thus

$$\Pr(S_n = S_{n-1} = \ldots = S_{n-i+1} = 0)$$

$$= \Pr(C_{n-i+1} = 0) \Pr(A_m = 0) \quad ; i = 2$$

$$= \Pr(C_{n-i+1} = 0) \Pr(A_m = 0) \prod_{j=n-i+1}^{m-1} \Pr(A_j = 0) \Pr(B_j = 0) \quad ; 2 < i \leq n$$

$$= \frac{1}{2^{i-2}} - \frac{(1 - R)}{2^{n+i-2}} \quad ; 2 \leq i \leq n$$

An i bit left shift will be required for normalization if there are exactly i leading zeros. Considering exactly one leading zero yields

$$P_1 = \Pr(S_n = 0, S_{n-1} = 1)$$
$$= \Pr(S_n = 0) - \Pr(S_n = S_{n-1} = 0)$$
$$= \frac{1}{2} \qquad \text{(6a)}$$

and for two or more leading zeros

$$P_i = \Pr(S_n = \ldots = S_{n-i+1} = 0, S_{n-i} = 1)$$

$$= \Pr(S_n = \ldots = S_{n-i+1} = 0)$$

$$- \Pr(S_n = \ldots = S_{n-i} = 0)$$

$$= \frac{3}{4}\left(\frac{1}{4}\right)^{i-1} - \frac{1-R}{2^{n+i-1}} \quad ; 2 \leq i < n \qquad (6b)$$

No shift is required when $S_n = 1$, or when the result is all zeros.
Hence

$$P_0 = \Pr(S_n = 1) + \Pr(S_n = \ldots = S_1 = 0)$$

$$= \frac{1}{4} + \frac{1-R}{2^n} + \frac{R}{2^{2n-2}} \qquad (6c)$$

Unlike signs, exponents differ by more than one

This case is very similar to the previous one, and can be analyzed by the same method. The bit layout after binary point alignment is shown in Figure 4. It can be seen that Equation 5 is applicable.

Only one leading zero can be produced, since to obtain a leading zero $A_{n-1} = 0$ and $C_{n-1} = 0$, and thus $S_{n-1} = B_{n-1} = 1$. Hence no more than a one bit left shift will be required for normalization.

If C_{m+1}, or any of $A_{n-1}, \ldots, A_{m+1}$, is a one then $C_n = 1$ and $S_n = 1$.

Therefore

$$P_1 = \Pr(S_n = 0) = \Pr(C_n = 0) = \Pr(A_{n-1}$$

$$= \ldots = A_{m+1} = 0) \Pr(C_{m+1} = 0)$$

$$= \frac{1}{2^{n-m-1}}\left(\frac{3}{4} - \frac{1-R}{2^{m+1}}\right)$$

$$= \frac{3}{4} \cdot \frac{1}{2^{s-1}} - \frac{1-R}{2^n}$$

$$; 2 \leq s \leq n \qquad (7a)$$

A shift of n bits can produce a leading zero when word

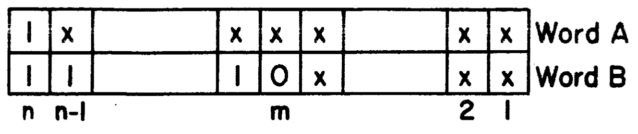


Figure 4—Numbers following binary point alignment and one's complementing of word B

B is rounded after the alignment shift, and the last (n − 1) bits of word A are zero.

Then

$$P_1 = \Pr(S_n = 0) = \frac{R}{2^{n-1}} \quad ; s = n \qquad (7b)$$

As with the case of like signs, if the exponents differ by at least n (n + 1 with rounding) no shifting will be required as the larger number is the result. Hence

$$P_1 = 0 \quad ; \quad s > n \qquad (7c)$$

As the only alternative to a one bit left shift is no shift

$$P_0 = 1 - P_1 \qquad (7d)$$

Application

In Table I is a tabulation of the probabilities given by Equations 1, 3, 4, 6 and 7 (assuming that 2^{-n} is negligible). As an example of how these probabilities can be used to optimize subprogram operation the addition of numbers with unlike signs will be considered.

Table I—Probability P_i of an i-bit normalization shift after an s-bit alignment shift

alignment shift s	P_i							
	like signs		unlike signs					
	P_{-1}	P_0	P_0	P_1	P_2	P_3	P_4	P_5
0	1	-	0	$\frac{1}{4}$	$\frac{5}{16}$	$\frac{13}{64}$	$\frac{29}{256}$	$\frac{61}{1024}$
1	$\frac{3}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{16}$	$\frac{3}{64}$	$\frac{3}{256}$	$\frac{3}{1024}$
2	$\frac{3}{8}$	$\frac{5}{8}$	$\frac{5}{8}$	$\frac{3}{8}$	–	–	–	–
3	$\frac{3}{16}$	$\frac{13}{16}$	$\frac{13}{16}$	$\frac{3}{16}$	–	–	–	–
4	$\frac{3}{32}$	$\frac{29}{23}$	$\frac{29}{32}$	$\frac{3}{32}$	–	–	–	–
5	$\frac{3}{64}$	$\frac{61}{64}$	$\frac{61}{64}$	$\frac{3}{64}$	–	–	–	–

For computers with a "shift-and-count" instruction for normalizing a number and counting the leading zeros a relatively standard subprogram is shown in Figure 5a.

All normalization is done using the "shift-and-count" instruction.

From Table I it is obvious that in many cases a one bit left shift would be enough to normalize the result, with a correspondingly simpler and faster exponent adjustment routine. In most machines, however, there is not a direct test for a single leading zero, and a programmed test loses any speed advantage that use of the one bit shift would gain for this special case.

For machines with a fast one bit shift Figure 5b presents an alternative philosophy: try a one bit shift and if unsuccessful proceed with the "shift-and-count." It is anticipated that enough time will be saved on single leading zero cases to compensate for the loss of time on the multi-leading zero cases.

To analyze the relative merits of the normalization schemes of Figure 5a and 5b define

r—time to process result with no leading zeros

a + bi—time to normalize a number with i leading zeros and process the result

c—time to shift left one bit and check if result is normalized

d—time to process result after successfully normalizing via a one bit left shift

For Figure 5a the average normalization time is

$$T_a = r\,P_0 + \sum_{i=1}^{n} (a + bi)P_i$$

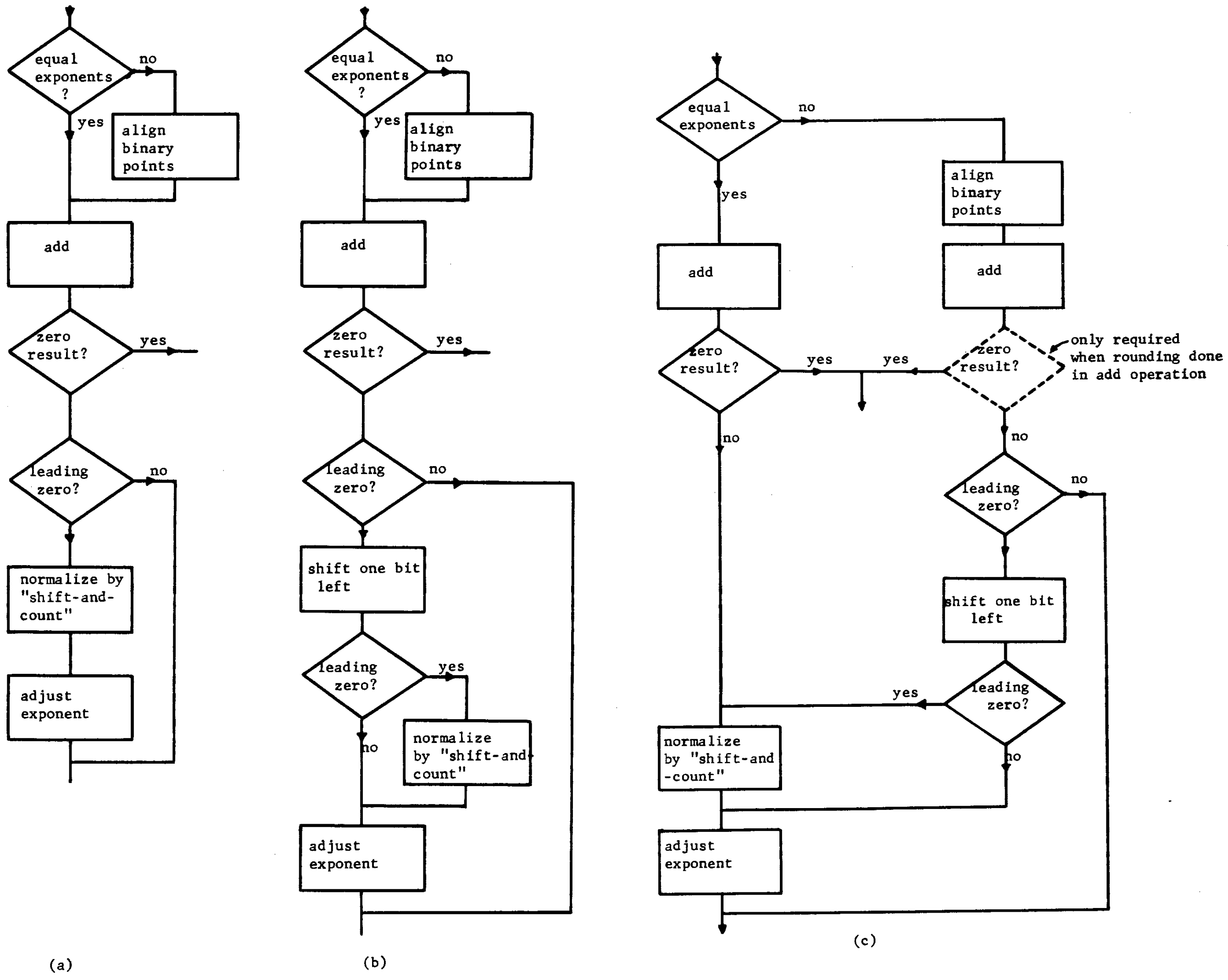


Figure 5—Possible subprograms for adding numbers with unlike signs

while for Figure 5b it is

$$T_b = rP_0 + (c + d)\,P_1 + \sum_{i=2}^{n} [c + a + b(i - 1)]P_i$$

$$= T_a + (c - b)(1 - P_0) + (d - a)P_1$$

It is evident that T_b may be greater or less than T_a depending on the machine characteristics controlling a, b, c and d.

For a floating point addition subroutine (with $n = 18$) for a PDP–9 computer it was found that $a = 15.6$, $b = 0.4$, $c = 4$ and $d = 7$. These produce the values for T_b shown in Table II. The second method is best for non-equal exponents but the first method is best for equal exponents. Since the information on whether the exponents were equal is available, the subprogram can be modified to the form shown in Figure 5c. This gives the advantages of Figure 5b to non-equal exponents, but retains the advantages, and improves on, Figure 5a for equal exponents. While an exact measure of the improvement in the normalization is impossible without the knowledge of the alignment shift distribution it would appear as if Figure 5c is about 3% better than Figure 5a.
Execution time of subprogram in Figure 5b.

Table II—Execution time of subprogram in Figure 5b.

alignment shift	T]
0	$T_a + 1.45$
1	$T_a - 1.60$
2 or more	$T_a - 5.00\,P_1$

SUMMARY

In the example above it is unlikely that the second method would have been considered if the table of probabilities indicating the high incidence of only one leading zero had not been available. Since the final subprogram is an improvement on the second method to eliminate a flaw detected during timing calculations, it is reasonable to assume that the best subprogram would not have been evolved without the use of normalization shift probabilities.

It must be remembered, however, that Figure 5c is the best for a particular machine. The only data that is directly applicable to another machine is the table of probabilities. It is still necessary for the designer to deduce a method where specific machine features may be exploited to reduce subprogram time. The table of probabilities allows him to check if the technique devised does yield the expected benefit of a faster program.

REFERENCES

1 D W SWEENEY
An analysis of floating-point addition
IBM Systems Journal Vol 4 No 1 1965 31–42

2 R W HAMMING
Numerical methods for scientists and engineers
McGraw-Hill New York 1962 37

APPENDIX A

Assuming equal probability of one or zero in all bits but the first implies a uniform distribution of numbers. However, Hamming[2] indicates that during floating point calculations numbers tend to move towards the lower end of the normalization range.

Using the Hamming distribution

$$P(x) = \frac{1}{x \ln 2}$$

then the probability that bit A_{n-j} equals one can be calculated by integrating the probability distribution over the range of numbers for which A_{n-j} is one.

$$\Pr(A_{n-j} = 1) = \sum_{i=0}^{2^{j-1}-1} \int_{1-i/2^j - 1/2^{j+1}}^{1-i/2^j} \frac{1}{x \ln 2}\,dx$$

$$= \frac{1}{\ln 2} \ln\left(\prod_{i=0}^{2^{j-1}-1} \frac{2^{j+1} - 2i}{2^{j+1} - 2i - 1}\right)$$

$$= \frac{1}{\ln 2} \ln \frac{2^{(2^j)}\,(2^j!)^3}{(2^{j+1}!)(2^{j-1}!)^2}$$

Using Stirling's formula

$$\ln x! = \ln \sqrt{2\pi} + (x + \tfrac{1}{2}) \ln x - x + \frac{\theta}{12x}$$

$$\text{where } 0 < \theta < 1\;;\; x > 0$$

the above expression reduces to

$$\Pr(A_{n-j} = 1) = \tfrac{1}{2} + \frac{2^{-j}}{12 \ln 2}\left(3\,\theta_1 - \frac{\theta_2}{2} - 4\,\theta_3\right)$$

$$= \tfrac{1}{2} + 2^{-j}\,\phi\;;\; -.541 < \phi < .361$$

Thus the probability converges to ½ as j increases. Table III shows the actual $\Pr(A_{n-j} = 1)$.

In view of the rapid convergence to a value of ½, and that the maximum deviation from ½ is not large, it seems reasonable to assume that ones and zeros are equally probable in all bit positions. Using the actual values from Table III for the first few bits would greatly complicate the model without significantly altering the result.

Table III—$\Pr(A_{n-j} = 1)$ assuming Hamming distribution

j	$\Pr(A_{n-j} = 1)$
1	0.415
2	0.456
3	0.478
4	0.489
5	0.494
6	0.497

A panel session—Software transferability

Program transferability

by JAMES A. WARD, *Chairman of Session*

Department of Defense
Washington, D. C.

During the early development of higher order languages in 1959, we were told that programs written in such languages could be run on almost any computer. Now, ten years later, we find that programs so written not only are non-transferable from one manufacturer's computer to that of another, but, in some instances, cannot be run on two computers of the same make and model with memories of different sizes. In the intervening years, computers have become much faster and the cost per operation has become much cheaper, while the cost of programming has become relatively far more expensive. Still, we do not have program transferability and millions of dollars are spent each year on the uninspiring task of reprogramming routines on additional computers.

The objection to transferability of software is that it is impractical: programs transferred would operate very inefficiently because of different internal machine organizations. Data is particularly difficult to transfer onto machines of different word length. If such transfers are made there is loss of efficiency. A system that would provide for such transfer would unduly restrict the freedom of the programmer and introduce inefficiencies.

This is why in many military systems all the computers are required to be identical, or in the same family, so that programs written for one computer can be used on all and that programs and data can be transferred. The current procurement for World-Wide Command and Control is just such a system. The lack of transferability also makes the upgrading of a computer a traumatic ordeal for almost any computer installation in government or industry. A number of people who have recently gone through this to obtain third generation equipment seem to think that the government (or DoD) should freeze present hardware designs to prevent reprogramming for the fourth generation of computers. They seem to forget that this is a perennial request. I was asked to join such a movement over six years ago and this request has been repeated every year thereafter. Had this movement been successful, the current third generation would not have come about.

The design of digital computer hardware has made tremendous strides during the past ten years. Further advancements are forthcoming throughout the field, particularly in military computers and scientific computers designed for large government problems. I feel it is criminal to stifle advancement; therefore, I do what I can to encourage new designs and internal organizations.

But we still need the compatibility among different machines that will permit the efficient transfer of programs, data and programmers from computer to computer without reprogramming. Since the hardware design should not be unduly restricted, the compatibility necessary for transfer must in large part be provided by software. I feel certain that this will eventually evolve. In fact, I am happy to report that progress toward solution is already on the way:

a. The Air Force at Rome Air Development Center has initiated an R and D program to determine requirements for software transferability.
b. Dr. Hopper, in the Navy, has developed a standard COBOL by which programs in that language have been compiled and executed on many different machines.
c. Mitre Corporation, under DOD contract, is studying the problem of transferability of data.
d. There are also several laboratories with different manufacturer's machines in which any program written can be compiled and executed on any machine.

I am convinced that such transferability of programs, data, and programmers is within the present state-of-the-art. This panel from government and industry has been assembled to tell what has been done and what is planned.

Let us all cooperate to hasten the day when most programs written in any higher order language can be compiled and executed on most existing computers. The time, money and manpower saved by eliminating reprogramming can then be used to solve other more interesting and useful problems.

Program transferability

by ROBERT W. BEMER

General Electric Company
Phoenix, Arizona

General

The problem of program transfer is such that most people think they understand the process better than they do. Optimism is rampant; success is elusive. I have some tenets which I believe to be sine qua non:

- Program transfer is complicated by each element which is different—user, CPU, configuration, operating system, etc.
- Programs must be planned for transfer. "After-the-fact" is virtually useless, like post-classification for information retrieval. The information loss is too high in the transfer from programmer to code. If everyone wrote and documented his program as a connectable black box, only the connecting process would need to be under the control of the user.
- In twelve years of hearing proponents discuss it, I have not yet seen successful mechanical translation of machine language programs. There are the processes which a translator:

    a. Thinks it can do and can.
    b. Thinks it can't do and says so, for human rework.
    c. Thinks it *can* do and *can't*, and therefore *doesn't say so!*

- Transfer should always be made on a source program basis. Recompilation is a trivial expense.
- To the highest possible degree, the documentation of the program should be self-contained in the source program itself (rather than in the auxiliary documentation), and in a standard format and placement so that mechanized program tools know where to find the machine-readable information for extraction and use.
- Production of identical answers is (particularly for scientific problems) an additional requirement which must be specified and paid for. Differences may be due in part to differing internal arithmetic modes, but more often they are due to the overlooking of imprecision in method. On balance, obtaining different answers must be considered a healthy phenomenon.
- The criterion which a software module/component must meet in order to be self-documented adequately is:

> "Can it be dropped into a program/data base for problem brokerage, whereupon a completely anonymous user may make a mechanical search to his requirements, find and use the module in his problem, and pay automatically a brokerage fee upon successful usage?"

This would be one standard that nobody would argue about—if he got found money at the end of the month, for conforming. Perhaps this might be a better solution than patenting software. Only thus can the non-specialist take advantage of computer utilities.

Some information required to transfer (run) a program[1]

- Program name (number)
  Program function
  Descriptors, classification (computing reviews)
- Original computer system
  Original configuration, subset of required configuration, options used/available
  Other systems/configurations verified to run on
- Operating system, requirement, linkages, interfaces
  Running instructions
  Store requirements (resident program, nonresident program, data, tables, segmentation, overlay sequences)
- Source language (standard, dialect)
- Input/output data
  Data structures
  Data types
  Data elements, collating sequence

(1) To complete while *producing* the program.

- Interfaces (other units called, libraries)
  Connections (via jumps, switches, natural flow)
  Languages/processors equipped to call this program
- Method, average runtime (for interactive simulators)
  Restrictions, constraints, degenerate cases, idiosyncrasies
  Range, accuracy, precision
  Changes occurring in conditions, status, original input
  Optional
  Information specific to program transfer
  Default options—referring to international/national standards
  Responsible organization
  Grade of program (thoroughness of testing)
  Test cases and answers (possible autoverification and answer match)
  Bibliography, references
  Copyright, price, etc.
  Source/object program listing, number of instructions/statements

Mechanical tools for conversion[2]

- Combinatorial path exercisers through a program
- Programs which page the source code for the programmer and mechanically force him to be up-to-date
- Programs which mechanically check the linkage of units of a software system to provide a directed graph for flow verification, ensuring that any software unit will not interface with other software units to which it should *not* be connected.
- Mechanical determination of valid paths in the reverse direction of flow, as a diagnostic tool for finding "How did we get here from there?"
- Mechanical verification of successful meeting of interface requirements when passing from one software unit to another in a forward direction.
- Mechanical re-verification of linkage and interface requirements for any revisions.
- Code acceptance filters.
- A patch defense (correct/change in source code only)
- (De-)flowcharters

Mechanical capture of facilitating information[3]

The source-to-object program translation process yields information. Much of this is lost, but needn't be. Some of this information concerns elements which are not themselves standardized, but can be part of a standard list of measurements useful to program transfer.

Therefore a language processor should be constructed:

- To be self-descriptive of its characteristics (i.e., features contained, added or missing; dialects or differences).
- To affix to the original source program, as a certification of a kind, either an identification of, or its actual characteristics. It may also strike characteristics or features which were unnecessary for that source program.
- To inspect transferred programs for a match to its own characteristics.

If the transferred program is processed successfully:

- The identification of the new processor is also affixed to the source program.
- In any area where the new processor has lesser requirements (i.e., a smaller table worked successfully; a missing feature was not required), the affixed information is modified to show the lesser requirement.

Thus a source program, once processed, contains information on:

- The minimum known characteristics required for successful processing.
- All processors (with operating systems) which treat the source program successfully.

(2) Used during the completion stage of the program, to prepare against transfer problems and to ensure a well-conditioned state.

(3) To obtain in each use of the program.

Software compatibility

by JOHN A. GOSDEN
The Mitre Corporation
McLean, Virginia

Data Exchange

There is a growing need for data exchange, particularly the passing of files of data between programs that were produced independently. This will be needed in the development of computer networks and data bases; for example, a head office installation collecting files from various divisions of a corporation to build a data base. Both the development of formal and informal computer networks as well as the economic feasibility of large data bases are favoring the development of arrangements for a considerable volume of data exchange, whether directly over communication systems or by the dispatch of reels of tape and boxes of cards. These are very significant areas of growth that are just beginning to emerge in commercial EDP and are already creating problems within the Federal government.

The development of data interchange is straightforward when the correspondents have agreed on the format, but where there has been no prior agreement, conversion usually involves considerable manual intervention. Some typical problems are that:

a. The sender's format may not be specified rigorously and an informal description may have to be debugged.
b. The sender's format may not be expressible in the receiver's system.
c. The sender's format descriptions may be embedded in the program.
d. The format in the sender's system may vary from record to record and be embedded in the data.

Any of these problems may arise when either an existing application is converted to a new system, or a new member of a cooperating network has a system different from that of any existing member.

There are two basic problems:

a. Few existing systems have any ability to deal with a new format automatically, and those that do are limited to data described in the same system.
b. The number of different, and often incompatible, ways of describing data is increasing; e.g., Format statements in FORTRAN, Data Description in COBOL, COMPOOL in JOVIAL, FFT's in FFS.

Any solution to this problem should not restrict participants in the use, within their own local system, of any internal data structure they like or any programming or query language they like. Therefore we need a standard data description language for data exchange. It is expected that systems should interface with a single way of describing data for interchange and provide conversion processes in their interfaces. If a suitable interface is to be developed, we will not want to standardize formats, which would be absurd, but we would want to standardize ways to describe formats. We also will want to attach the data descriptions to the data, so that the transmission of both data and its description can be performed without manual intervention.

A data description language for data interchange does not have to be read and understood principally by humans. It can be thought of as a complicated coding to be generated and interpreted by the interface modules of systems in a network. In a well-designed system a user would describe data in the form provided for local use, and the system would translate to data interchange conventions. Therefore, the data description language should be generally compatible with data descriptions in current programming languages. Later, developments in programming languages may be influenced by a desire to remain compatible with data interchange conventions.

Standardization of high-level languages

by GRACE MURRAY HOPPER

Director, Navy Programming Languages Group
Washington, D. C.

The terms "compatibility", commonality," and "transferability" are used in discussing the mobility of programs and programmers. The common element essential to such mobility is the establishment of standards for programs, programmers, and documentation. The adoption of high-level programming languages such as COBOL, FORTRAN, ALGOL, and JOVIAL is a required element of such standards. The high-level languages are innately self-documenting—an essential for transferability. Thus, their use provides assistance in the transfer of programs among activities; the conversion from one computer generation to another; the conversion from one computer manufacturer to another; and the transfer of programs for back-up and readiness.

Further, the programmers, themselves, need be trained but once and retraining upon transfer to a new system is virtually eliminated. Programmers become capable of greater productivity per unit of time resulting in less cost per unit of program. However, such application of the programming languages implies that they themselves be standardized. A standardized language must be commonly acceptable, competitive, developable, maintainable, and useful. The standard language must be developed, defined, approved, adopted, and installed.

The installation of a standard programming language, in varying environments with varied requirements, requires of management the normal functions of promulgation, installation, control, evaluation, monitoring, and provision for maintenance and changes. The execution of these functions can be assisted by the use of programmed tools such as validation routines, translators, preprocessors, flowcharters, debugging aids, and by standard manuals and instruction courses.

The discussion includes a survey of the growth of the standard languages with implications and suggestions for future developments. Information supporting the need for the application of the languages and the resultant time and cost savings is introduced. The necessary components of an installation package are defined and their implementation discussed. The need for management interest, concern, support, and action is stressed.

The transferability of computer programs and the data on which they operate

by EDWARD MORENOFF

Rome Air Development Center
Griffiss AFB, New York

INTRODUCTION

Software transferability involves the transfer of programs and the data on which they operate from one arbitrary operating environment to another, with the expenditure of only a small fraction of the initial programming development time and cost. The programs can range from small routines for evaluating trigonometric functions, to large and complex systems such as compilers, data management systems, or command and control systems. The environments in which these programs are to be executed may be either slightly or highly dissimilar with respect to machines, machine configurations, or operating systems and languages used.

The interest of the Rome Air Development Center in this area extends over a period of several years. During this time, both hardware and software research and development programs have been initiated which either directly or indirectly contribute to the solution of the problem. The utility of adopting standards, and the form these standards might assume, were defined in 1967.[1]

Typical of the efforts in the hardware area was a study leading to the definition of a microprogrammed computer main frame capable of efficiently executing a number of different machine language instruction sets.[2] Typical of the efforts in the software area was the work leading to the design of a generalized data management system,[3,4] the development of modular proramming design tools,[5,6] and the Information Processing Code.[7]

In January 1968, the Program Transferability Study Group* was established. Its principal objective was to examine the whole area of software transferability formally, and see what, if anything, could be done to eliminate the problems associated with transferring programs and data between arbitrary environments.

The preliminary findings of the study group were released in June 1968.[8] The Group found the main obstacles to software transferability to be loose specification of data structures, lack of programming standardization, and lack of freedom when higher level languages were used. Possible solutions to these problems are: (1) Administrative control of programming and documentation (2) Extensions to current languages (3) Use of a new programming environment which would eliminate the constraints of the older system.

The problem

The study group concluded that the problem of transferring a program between arbitrary operating environments was not solved by the current technology, even when the initial program development made use of a standard version of a single higher level language. Current practice is such as to require changes to the initial form of the program itself rather than simply a recompilation, in order to adapt the program to a change in environment.

One of the principal factors necessitating such changes is the lack of adequate facilities for the explicit specification of data, programs, actions, messages, linkage, and the like. The programmer is instead encouraged, if not actually forced, to make implicit in the form of his program many of details of its initial environment and the data to be operated on. In order to transfer a program from one environment to another, reprogramming is required to the extent that differences in the two environments must be reflected in changes to the program.

Closely related to the lack of adequate explicit specification facilities is the lack of constraints which would serve to regularize the behavior of the programmer. As a consequence of the excessive generality and complexity

*The Program Transferability Study Group was chaired by George H. Mealy. Other members included T. E. Cheatam, Jr., David J. Farber, Edward Morenoff and Kirk Sattley.

in present operating systems and languages, a programmer setting out to prepare a complete program has entirely too much freedom. In the absence of explicit facilities or when such facilities are too general, different programmers produce entirely different programmatic solutions to the same problem. A strong relationship is believed to exist between this difference and the difficulty in transferring programs.

An example of a language which has enjoyed a measure of success in the development of transferrable programs is COBOL. COBOL encourages the explicit description of data rather than the implicit description inherent in most other languages. It is about the only language system which permits any kind of environment specification. It is limited in scope, however, to problems which trace their origin to unit record equipment. This narrowness of scope appears to be one of the principal reasons for its success in allowing programs written in it to be transferred between environments.

In addition to the use of higher level languages, the study group found two other basic approaches to solving the transferability problem, both largely unsuccessful. The first approach involves "decompiling" a binary, decimal or symbolic program written for one operating environment, and then generating new code for some other operating environment. The second approach is by the application of a set of management solutions. These solutions range from the insistence on replicating functionally identical hardware configurations to setting standard specifications on programs so they are completely modular with very precise functional specifications and clear interfaces.

Attacks on the problem

The second major conclusion of the study group was that the current state of the computer technology had reached a level of development at which something could be done about the problem of transferability today. Indeed, three compelmentary levels of attack were proposed.

The first level of attack, designated as level A, deals with the definition and application of suitable administrative controls, both to the programming and documentation processes, using languages and operating systems as they exist today. The resulting standards would form a common subset or intersection of the capabilities of the operating systems and languages to be involved in the family of transferrable environments. To work at all, these standards must be enforced by a czar with final approval over all programming work.

The second level of attack, designated as level B, deals with the development of extensions to the current language and system base. This involves identifying the deficiencies with respect to particular existing systems, defining how these deficiencies may be eliminated, and then implementing the resulting changes to equalize the capability of all the operating systems and languages in the family of transferrable environments.

Part of the design problem at level B is to identify which extensions can be incorporated into the existing systems with a high probability of success within a two year period from the time they have been defined. Extensions not falling within this category would be deferred for consideration as part of attack level C.

The third level of attack, designated as level C, relaxes the requirement of using the existing base of operating systems and languages to allow consideration of a wholly new programming and operating environment which would substantially eliminate, for most practical cases of interest, the problem of program transferability. Within this environment the higher risk and/or longer term concepts would be tested and evaluated.

The life of the study group has been extended. It is currently continuing its investigations of selected aspects of the software transferability problem.

BIBLIOGRAPHY

1 E MORENOFF J B McLEAN
An approach to standardizing computer systems
Proc 22nd National ACM Conference Thompson Book Co August 1967 527-536

2 DECISION SYSTEMS INC
Interim Status Report No 1
Contract No F30602-68-0223 August 11 1968

3 P J DIXON J D SABLE
DM-1 a generalized data management system
Proc S J C C Thompson Book Co April 1967 185-198

4 E MORENOFF J B McLEAN
On the design of a general purpose data management system
Proc Fourth Annual Colloquium on Information Retrieval Int Information Inc May 1967 19-30

5 E MORENOFF J B McLEAN
Inter-program communications program string structures and buffer files
Proc S J C C Thompson Book Co April 1967 175-184

6 E MORENOFF J B McLEAN
Program string structures and modular programming
Proc National Symposium on Modular Programming Information and Systems Press June 1968 176-186

7 E MORENOFF J B McLEAN
A code for non-numeric processing applications in on-line systems
Communications of the ACM Vol 10 No 1 January 1967 19-22

8 G H MEALY et al
Transferability study group
Rome Air Development Center Technical Report No TR-68-341, December 1968.

Transferability of data and programs between computer systems

by JEROME D. SABLE

Auerbach Corporation
Philadelphia, Pennsylvania

INTRODUCTION

The cost of problem analysis and programming dominates by far the cost of computer utilization. This dominance will increase in the future as hardware elements become more powerful and economical and problems become more complex and demanding of highly skilled analysis. As this trend continues, the sophisticated user will become increasingly intolerant of a situation which prevents economic transfer of programs and data from one installation or hardware type to another. A standard machine independent environment which provides data management services at several levels to programmers and task programs should be defined. Programs would then interface with a set of virtual machines, or standard program environments, which are independent of the particular hardware configuration of the installation.

The *traditional* approaches to data and program transferability have been through (a) the use of compatible hardware types which have presented "equivalent" hardware interpreters for data and program, and (b) the use of standard higher level procedure-oriented languages and compilers to translate programs to a particular machine type. The first approch guarantees complete interchangeability only as long as the program's support software is duplicated but removes the possibility of matching special hardware types to problem areas for which they may be particularly appropriate. This restriction is unnecessarily severe in many cases. The second approach avoids this restriction but there are many problem areas for which standard procedure-oriented languages have not been adopted. Indeed, many systems will continue to be implemented in assembler and macro level languages.

I would like to suggest that the problem of data and program transferability can be approached most generally by extending approach "(a)" to include software as well as hardware interpreters. Software interpreters, or simulators, have been used in the past to transfer a machine language program from machine A to machine B. However, these have not been entirely successful or widely used. The success of this approach hinges on the ability to write programs for one of several standard environments and to describe in a standard, yet general, way the data structures which are to be transmitted and interpreted. To permit general applicability from machine level programs and data, through to higher level language programs and other character stream messages, requires that a wide range of data structures and languages be describable in a standard way. The language description standards must include the lexicographic, syntactic, and semantic levels.

In the following paragraphs a hierarchy of data structure types will be described which range from machine and storage-oriented structures to "logical" data structures transmittable as character strings independent of physical representation. They are offered as one possible approach to a comprehensive set of data structures.

Data structures

The cell

The most primitive concept to be considered is that of an address space. This is viewed as a region of atomic elements or cells which are addressable with some address word. There may be a hierarchy of cells such that higher level cells form an address space for a lower level cell. The cell is viewed as a region is which data can be stored and accessed rather than the data themselves. Several address spaces, or stores, may be involved in a system or network.

A given computer will have a system of primary, secondary, tertiary, etc., stores associated with it. A cell in any store is addressable with an address or a simple transformation of an address.

The truck and train

The most primitive relocatable data structure is the truck. This is a module of data which can be stored in a cell. A sequence of trucks, called a train, may be defined and transmitted from one store to another. Any relocatable data structure (the term as used here includes programs) is ultimately handled as a train whose trucks are bytes, words, or pages.

The bead, strip, and plex

An element (e.g., train) of fixed length and defined field structure will be called a bead. A sequence of beads of the same type will be called a strip. If the beads contain fields which address other beads then a network of beads, called a plex, is formed.*

*The terms *plex* and *bead* have been used by D. T. Ross in his work on the AED systems.

The stream

A list or list structure of byte trains of various lengths will be called a stream. Consider the data structures necessary for representing a process, or active job. These include a stream containing procedures and subroutines, a stream for the dynamic working data (or activation records) of each procedure, and a stream for the control stack which provides subroutine control. Collectively, then the process can be considered to be a stream list.

The item

Finally, a data structure may be defined in terms of named entities, called items, which bear a defined relationship to each other. These items are fields, records, files, and statements and are independent of any storage structure or address space. Items may be independent messages or form a highly structured hierarchy of nested elements, files, etc.

Specification languages

The success of the approach to data and program transferability being proposed here depends on the adoption of a comprehensive data description language (DDL). The DDL must meet a number of requirements:

1. It must be able to handle at least the range of data types listed above.
2. The requirement to handle item structures and source language messages means that it should be capable of expressing the symbols and syntax of context-free languages.
3. It should be capable of expressing the semantics of translation or interpretation of the source language strings, including procedure calls to other processors.
4. It should be representable in a graphical format that exhibits the structure of the language or data being described so that it represents an effective tool for *human* communication.
5. It should also be representable by linear strings amenable to transmission and computer input, and it should be easy to translate from the graphical to the linear version, and vice versa.

CONCLUSION

The problem of transferring data from one computer to another, and interpreting the data correctly, can be approached as a problem of constructing an adequate range of data structure types and devising a standard way of describing these types. No one level of data description is adequate. Rather, there must be a range of structures which go from the highly machine oriented cell structure to the user-information oriented structure. It is felt that a hierarchy of data structure types which is adequate for this task can be devised. The data types to be included have been listed above.

The problem of data description, however, is but a special case of language specification and special attention must be paid to the lexicographic, syntactic, and semantic aspects of specification. Some recently developed language specification languages and generalized language processors can be brought to bear on this problem. These processors can be employed as a standard interface in a network of different computer types. When furnished with the description of the structure (languages) to be accepted, they can carry out the appropriate translation and interpretation.

The stratification of data management servicesi nto a number of standard levels would make it appear to the programmer, that at any one moment, he is interfacing with one of a number of virtual machines which form an upward compatable hierarchy. These services may be, in fact, provided by a mix of hardware and software modules which depends on the particular system implementation, and the hardware types being used in a given instance. Thus, each program has, as its interpreter, a virtual machine whose interface with the program is know but whose composition may vary and is irrelevant except for timing considerations. A given installation, because of hardware and software modules used, may provide interpretive virtual machines only up to a given level, requiring programs written for a higher level virtual machine to undergo a translation process down to the appropriate level before interpretation can take place.

That the goals outlined above will be difficult to achieve in today's technology and market place is well recognized. However, I feel it is the responsibility of the industry to its users to embark on a comprehensive project to ensure software transferability across a wide range of problem areas and hardware types without limiting the development of new languages and machines.

A panel session—Computer-assisted instruction: Current status—Future problems

PATRICK SUPPES, *Chairman of Session*

Stanford University
Stanford, California

CAI problems and prospects

by WALLACE FEURZEIG

Bolt, Beranek and Newman Inc.
Cambridge, Massachusetts

and

SEYMOUR PAPERT

Massachusetts Institute of Technology
Cambridge, Massachusetts

The expression "computer-assisted instruction" (CAI) is generally used to describe situations in which the computer is used as a teaching surrogate in some sense—whether as drill instructor, tester, or specialized tutor. Most applications of this kind have had specific, limited, and modest educational goals. When used to administer drills or branching tests, a computer is not called upon to be intelligent—only useful. Yet it is interesting and important to ask whether the computer can become an intelligent artificial teacher, and more generally, whether there are valuable ways of using computers for teaching and learning.

In artificial teaching, the computer controls the interaction with the student. There are applications of the opposite kind, where the student controls the machine. The most common one is the teaching of computer programming itself. Another is the use of a computer to simulate a "real" laboratory. And, there is a potentially rich spectrum of intermediate arrangements—strong instructional interactions—in which the student and the computer share control and direct each other. As an example, student programming and artificial teaching might be coupled by having the computer monitor a student's work as he uses a programming language to perform a simulated experiment or to solve a problem. No significant experiments in this direction can be done without a great deal of work; but we do know, in principle, how to make a program follow the steps of a student who is not constrained by a stereotyped pattern and how to diagnose his difficulties on the way.

We shall argue that computers will make deep contributions to education in all three areas:

1. first, and with capabilities already well established, through the teaching of programming languages;
2. ultimately, and to an extent largely dependent on progress in artificial intelligence research, as an artificial teacher;
3. intermediately, as an instructional monitor or assistant, in a number of different subjects as diverse as music, language, and physics.

Along the way, we shall elaborate on specific educational contributions including the following.

1. The teaching of programming can provide a conceptual and operational framework for the teaching of mathematics.
2. Using an appropriate language, programming can be introduced routinely to third-graders for its special value in teaching the skills of clear and precise thinking and expression.
3. The computer can enhance the teaching of "practical" subjects (such as navigation or speaking a foreign language) whose mastery requires the integration of mechanical and intellectual skills.

Finally, we shall contrast the present lack of depth and perspective characterizing much of the work in this field with its rich prospects. In particular, we shall discuss our view that a serious investigation of the problems involved in developing an intelligent teaching system will yield rich results in the fields of computers, education, and psychology.

CAI: Research requirements for instructional strategies

by DUNCAN N. HANSEN

Florida State University
Tallahassee, Florida

The current developments of computer-assisted instruction (CAI) can be characterized as a phased transition from the creation of hardware and language systems for implementation into the more fundamental examination of the features of optimal instructional strategies for CAI applications. Instructional strategies are the plans by which informational presentations are matched with the current requirements of a learner in order to optimize on a set of criterion objectives. The research approaches into the nature and process of instructional strategies has been twofold, namely, naturalistic and systematic.

The naturalistic approach consists of three applicational types that developed concurrently with the hardware and language CAI systems. First, the complementary CAI type provides instruction that is adjusted to the stage of progress that a student has acquired in the conventional educational classroom; this approach is best exemplified by the Stanford Drill and Practice Mathematics Project. Secondly, the autonomous CAI type provides instruction that is the full corpus of an accredited course; the physics course at Florida State University and the library science course at the University of Illinois best represent the autonomous type. Lastly, the enriched CAI type provides instruction on content considered as extensions of the conventional classroom curriculum; the simulation games at BOCES are excellent examples of the enrichment type. An analysis and comparison of the instructional strategies of these three naturalistic CAI applicational types will be discussed.

The systematic approach to CAI research on instructional strategies resolve into six areas. First, what is the appropriate media for the presentation of a given concept? Second, what are the desirable time parameters for the CAI system to respond to the learner's answer? Third, what are the characteristics of the answer analysis routines that promote the specified learning objectives? Fourth, what is the decision logic of the presentation plan that specifies the sequence, amount of practice, and termination of the instruction? Fifth, what are the payoffs to the instructional process within CAI? And lastly, what kinds and types of reports should be part of CAI application? Current research findings and implications for future investigations are discussed within the framework of the six aspects of instructional strategies for CAI.

Instructional uses of computers to grow gracefully and effectively

by ELDO C. KOENIG

University of Wisconsin
Madison, Wisconsin

INTRODUCTION

In this brief discussion, it is suggested that three principal functions be recognized and be performed in parallel in order to achieve a graceful and effective growth in the instructional uses of computers. These functions are described as:

1. the practice of teaching using computers as aids;
2. research and development directed toward the practical uses of computers in teaching and learning;
3. basic research in intelligent systems.

Function (1) is dependent on function (2) and (2) is dependent on (3). Computer hardware and software and people are included in the discussion of each of the functions.

The practice of teaching using computers as aids

Computers in education today can be used most effectively to assist in the teaching of subjects that are mathematically disciplined, such as, programming, mathematics, engineering, physics, statistics, and bookkeeping. For courses of this type, much of the knowledge is contained in mathematical expressions and communication between student and computer can be more easily accomplished than in verbally oriented courses. Computers used in courses of this type can (1) assist individual students in a direct learning experience, (2) perform the calculations in solving assigned problems, and (3) aid the instructor in demonstrating the functional characteristics of the various types of mathematical equations, the effects of changes in the variables of computational models, and various physical phenomena through simulations.

Hardware and Software. The function should employ existing computer systems. The one selected must be a practical operating system with integrated hardware and software and with well-defined capabilities. Because rapid advances will continue to be made in systems, the policy should be that of leasing. Currently, the choice can be made between the leasing of terminals of a large utility type system and the leasing of a relatively small complete system. Small computer systems are available with complex man-machine interfaces which make them very capable in special types of applications, such as, graphics. The leased terminals of large utility systems are relatively simple man-machine interfaces, and the systems have limited capabilities in a conversational mode of operation. Since the fixed cost of leasing is low and the usage cost is variable, the number of students and the amount of time each uses a system can be small to accommodate a minimal budget. Because a leased complete system has a fixed total cost, it may be difficult to find enough student users to make the cost per student a reasonable value.

People. If current systems can be used effectively as aids in teaching mathematically disciplined subjects ranging from arithmetic in elementary schools to advanced college subjects, and if they can be cost justified for small numbers of students, then the only additional requirement is the training of teachers in the rules for communicating with the computers and in methods for using them. Usually, courses are offered for training in the rules of communication. Until similar courses exist for training in the new methods, teachers should be encouraged to experiment in developing simpler methods and techniques through the use of leased terminals of systems of limited capabilities. They will gain valuable experience which will prepare them for future systems of greater capabilities.

Research and development directed toward the practical uses of computers in teaching and learning

This function establishes teaching methods using computers as aids, determines the psychological factors related to these methods, and provides for the training of teachers in practical application of these methods, particularly those teachers in the lower levels of education.

Hardware and Software. An integrated hardware-software system for this function should be supplied as a practical result of research in intelligent systems, the third function, and should be replaced every three to five years with one of greater capabilities. The packaged system should provide for the maximum of flexibility and should be able to:

a. gather and analyze physiological information on a human subject
b. receive and analyze the various requested responses of the subject
c. provide flexibility in selection of the content and organization of learning materials and the interface devices
d. provide a variety of learning strategies through a choice of (b), (c), and an alternating sequence or an integration of the two
e. gather information on the environment of the subject
f. permit a higher level of control which coordinates the controlling actions for (a), (b), and (c)
g. permit a high level supervisory control with flexibility for coordinating and directing two or more subjects at different terminals in competitive learning experiments.

People. There should be a number of groups in different fields in a university performing this function, for example, the School of Education, Engineering, and Computer Sciences. The School of Education should place emphasis on the lower levels of education. Special training may be required in computational modeling, computers, and in the capabilities and the use of the man-machine interactive system to be used.

Basic research in intelligent systems

This research, in the broadest sense, should strive to use the human intelligence of a society better by improved ways of synthesizing intelligent systems and by more effective ways of transferring knowledge from one generation to the next. It should also strive to increase the intelligent capabilities of a society by imparting intelligence to machines.

Hardware and Software. Current hardware should be selected and developed into a hardware system, and initial software design should be based on the principles determined from the previous research system. Basic research should then be applied to the current machine system to improve its performance as a component of an intelligent system. The improved system, in turn, should further basic research which is again applied to produce additional improvements in performance. A machine system with significant improvements should evolve every three to five years and then should replace the system in research and development directed toward the practical uses of computers in teaching and learning.

People. Since complete interactive systems contain both man and machine as components, the research must draw on select basic knowledges from a number of fields. Unifying disciplines must evolve relating the various knowledges, and a graduate program should be associated with the research effort. The students must be able to acquire the variety of existing knowledges, and through their active participation in research, they should have an opportunity to apply and associate these knowledges. The build-up of both the research effort and the graduate program must be gradual.

A picture is worth a thousand words—and it costs . . .*

by J. C. R. LICKLIDER

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

The present is a critical, frustrating, and exciting time in the history of computer graphics. A dozen significant trends are developing in the same field of view. Several much-needed trends are not developing. The technology is changing so rapidly that four generations of graphical systems are in operation at once. "Computer graphics" means different things to different people. This is therefore a difficult session to "introduce."

Perhaps the most important introductory remark to make about the session is that the papers that follow this one do not attempt to cover the topic of computer graphics. They are *in* computer graphics, not *on* it. In this paper, I shall not try to cover the topic, either, but I shall take a brief look at the over-all field and try to sketch out enough of a map to provide an orientation and context for the papers that follow. In the process, I shall unburden myself of a few deep convictions.

There are three main efforts in computer graphics. The first is to improve the capability of graphical displays to represent things and processes. This capability extends into dimensionality, verisimilitude, complexity, and motion.

The second main effort is to improve the interaction between men and computers through graphics, to improve graphics as a medium of communication. It is important to distinguish between communication and representation. Both involve languages, but not in the same way.

The third main effort is to develop applications of computer graphics. Applications run the gamut from computer-aided composition of printed pages to computer-aided design and dynamic modeling.

All the foregoing pose problems of computer-system design and are energized and/or inhibited by economics.

This paper will deal, then, in an introductory—and necessarily sketchy—way with representational capability, communication and interaction, applications, computer-system design, and economics—all in the context of computer graphics. Because the art is long and the time fleeting, the paper will consist essentially of observations and assertions within those areas rather than reviews or analyses of them.

Representational capability

Although most computer graphics is limited to arrangements of points and line segments, together with perhaps a hundred elementary figures (characters), computers are of course capable of processing and displaying areal pictures replete with graded lightness (brightness) and varied hue (color). The work of Stockham[1] at the M.I.T. Lincoln Laboratory and the University of Utah illustrates the versatility of the computer in handling still pictures with gray scale. Stockham scans a low-contrast photograph into the computer memory, takes the logarithm of every lightness coefficient, applies a la Ernst Mach a second-spatial-derivative operator to each neighborhood of coefficients to yield a new coefficient for the central element of the neighborhood, then takes the antilogarithm and displays the resulting picture. Detail not visible in the original stands out clearly in the processed picture. But the processing requires many seconds for a picture with a millon elements, and Stockham's kind of picture processing is not kinematic or "on-line interactive" in the present state of the technology. Nor does the computer "understand" the

* The ideas described herein were derived mainly from research supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

picture in the sense of forming a structured model of the pictured scene in which the various objects and their parts are represented separately in a hierarchy. Thus in Stockham's application we see an emphasis of high precision in representation and processing at a sacrifice of speed and structuredness.

The work of Evans, Warnock,[2] and their colleagues at Utah starts with a structured model in the memory of the computer, rather than a high-information-content photograph at the input scanner. They have developed methods of processing that yield a halftone-like picture, in color, with "hidden lines" suppressed—in a small fraction of the computer time that would have been required two years ago. At the present time, however, even with the Utah methods, one cannot process and display areal (e.g., halftone) pictures fast enough to produce moving pictures in real time. He can go to slow-time frame-by-frame generation of film, or he can schematize, or both.

The technology for handling schematized pictures and diagrams—made up of points, line segments, and characters—has advanced markedly during the last two years. Roberts[3] described some of the basic ideas, involving homogeneous-matrix representation and perspective transformation. In the work that won them the prize at the Fall Joint Computer Conference, Robert Sproull and Ivan Sutherland[4] developed hardware that very rapidly transforms selected parts of a model in the computer's memory to a line drawing on the cathode-ray screen. The hardware rejects all the parts of the model—a large two-dimensional drawing or a three-dimensional scene—that would not be seen through a given window, and it produces the magnifications or perspective transformations and the clippings or trimmings at the window frame necessary to display just what would be seen through the window. It does all that fast enough to display 3000 lines flicker-free. The "3000 lines" refers to what is actually displayed; there can be many more lines that that in the model. The hardware handles curved lines and surfaces. It works hand-in-glove with the Warnock hidden-line algorithm, which works on schematized (linear) as well as areal displays, and the hardware is applicable to, and will decrease the processing time required for, areal displays.

The foregoing dealt with actually two-dimensional displays of two- and three-dimensional objects and scenes. There are interesting ways to make the displays perceptually, or even actually, three-dimensional. The method of calculating and presenting two slightly different "stereo" views is well known. So are methods based on the "kinetic depth effect," which present a single, changing view. When successive two-dimensional cross sectional images are projected in rapid sequence upon a moving screen, the eye, because of its persistence, sees the composite three-dimensional image. See Traub.[5] The computer can control the settings of a matrix of push rods to make a three-dimensional surface, a histogram rising from a plane. And, of course, it can synthesize a hologram.

Of those and other approaches to 3-D, the most exotic appears to be one demonstrated by Ivan Sutherland and associates at Harvard last year. The computer has a model of a situation in its memory, and it receives signals from sensors that tell it where the observer is and in what direction his head is oriented. With the aid of hardware described earlier, the computer then displays the part of the modeled situation that the observer would see. The display is projected from a small cathode-ray tube on the observer's head through a semi-silvered mirror into one of the observer's eyes, and the observer sees the displayed part of the situation out in front of him. He can look around and walk around—and always see what he would see if he were in the modeled situation. When I saw the demonstration, the hardware was not finished, and the situation was just an outline room with windows, a door, and a geometrical piece of statuary. Even at that, it was quite an adventure. Given situations defined by thousands of line segments, one could surely create some exciting experiences—and probably some very significant ones. As Sutherland has pointed out, the "physical" laws of the modeled world can be determined by the programmer. They can change with time. They can even depend upon the observer's behavior. Elsewhere, the possibility of creating a direct four-dimensional experience has been discussed. See Licklider.[6] I shall not go through it again here, but let me say that it will be intellectually at least as exciting to perceive and explore a synthetic 4-D world as to perceive and explore a merely actual, merely 3-D moon.

Most of the capabilities of representation described in the foregoing are characteristic at present solely of expensive equipment and are regarded by many as esoteric. I shall say something about that under the heading, *Economics*, but for the moment let it direct our attention to the other end of the spectrum. Among the most significant developments as judged from a practical point of view, certainly, are computer setting of type, computer-aided preparation of diagrams and graphs for the printed page, computer-aided composition of advertisements, computer-aided generation of films, off-line production of graphics (as with curve plotters and as with the Stromberg-Carlson 4020), and fast on-line display of alphanumeric data and text.

Communication and interaction

It is of course very important in computer graphics to be able to represent things and processes accurately and in detail. I think it is more important to be able to represent them in proper structural organization. I think it is still more important for computer graphics to be an effective medium of communication. Good one-way communication is good, but good two-way communication—good graphical man-computer interaction—is better.

Now, as the history of art shows, and as Huggins and Entwisle[7] will emphasize, and as I hoped Don Hatfield would develop in depth in this session, the representation with the greatest verisimilitude is not often the best means or medium of communication. Good graphical communication is more a matter of linguistic expression than it is of pictorial reproduction. Unfortunately, we in the computer world do not know much about the language, and unfortunately not many of us are trying hard to find out about it. Obviously, it does not have much to to with programming languages. It probably does have something to do with data structures. But mainly it is the art of expressing ideas through configurations and manipulations of signs and symbols. Let me call that art an essential extension of computer graphics and then leave the topic to Huggins and Entwisle. I like their approach to it, insofar as the one-way language—computer to man—is concerned.

In my assessment, however, communication is essentially a two-way process, and in my scale of values, interaction predominates over detail, gray scale, color, and even motion. In my judgment, the most important problem in computer graphics is that of establishing excellent interaction—excellent two-way man-computer communication—in a language that recognizes, not only points, lines, triangles, squares, circles, rings, plexes, and three-way associations, but also such ideas as force, flow, field, cause, effect, hierarchy, probability, transformation, and randomness. Nowhere, to the best of my knowledge, is such interaction approached in a broad problem area at the present time. The nearest things to exceptions are the graphical computer-programming interactions of Ellis and Sibley[8] at RAND, the partly graphical, partly alphanumeric on-line augmentation of Engelbart's[9] intellect at Stanford Research Institute, and the graphical explorations in architecture of Negroponte[10] at M.I.T. It is very frustrating to me that five and a half years have elapsed since Sketchpad passed its milepost without bringing more progress in man-computer interaction at the level of ideas and concepts.

It seems unlikely that work in computer-aided design of devices and structures will lead to the advances I am hoping for. Computer-aided design of complex systems or processes might do it. I like the name, "interactive dynamic modeling," for the art. I want to see develop a kind of combination of computer-aided design á la Sketchpad and computer-program simulation á la on-line GPSS or OPS—in which the modeler and the computer engage in a high-level graphical interaction to formulate and test hypotheses for the solution of difficult problems that are not amenable to straightforward logico-mathematical formulation, that involve more synthesis than analysis, more discovery than proof. Well, it is not much to have the name. I wish I had the method, the language, the software, and the hardware.

Man-computer interaction of course presupposes computer-input devices (as well as languages) to mediate the communication from man to computer. The man should be able to signal the computer in a natural and synergic way, as by pointing and marking with a stylus while enunciating control words or phrases. He should be able to write or draw on the same surface as the computer. I shall not develop this part of the discussion beyond saying that one can appreciate the problem after he has read the transcript of a good blackboard lecture and then later (and separately) seen photographs of the blackboard. "Chalk plus talk" is an excellent medium. Neither chalk nor talk alone carries much information. The communication must therefore lie in the coordination.

Applications

At present, it appears that three very different classes of application of computer graphics are successful: (1) routine applications in the field of publishing (e.g., *Chemical Abstracts*) that yield printed pages as primary output and at the same time retain the information in computer-processible form for updating or secondary exploitation; (2) routine applications throughout business and industry (e.g., airline ticketing, display of stock quotations) that involve fast on-line display of alphanumeric data or text from computers; and (3) "cream" (and often highly nonroutine) applications in government and industry (e.g., military command and control, space-mission control, seismic prospecting for oil) in which the premium on being first or best is very high or in which the required results cannot be achieved any other way. The first two do not require sophisticated graphic capability. The third requires and can afford very sophisticated capability.

If and when there exists and is available the kind of computer graphics I referred to as "interactive dynamic modeling," computer graphics will become a part of thinking and problem solving and decision making wherever those functions are carried out. Design will of course be a major application area: design of all kinds of systems and processes—space, urban, transportation, manufacturing, military—as well as devices and structures. Management of government and business will use graphics in its "command and control." Applications will abound in research and development and in medicine. But the application area *par excellence* will be education. Almost anything not involving muscular skill that needs to be explained and demonstrated can be explained and demonstrated best with the aid of interactive computer graphics. For the sake of brevity, I shall leave that as a simple assertion in need of proof by demonstration.

The trouble is, all those applications that depend upon a significant augmentation of the human intellect (to use Engelbart's phrase) demand a level of computer graphics as sophisticated as that required for the "cream" applications mentioned earlier—or even more sophisticated.

The field is ready for and cultivating simpleminded applications in which computer graphics will do faster or less expensively things that can already be done without it, but the field is not yet ready to accept the challenge of "mind expansion." That carries us to the economics again—which we shall examine very shortly.

System design

The prevailing idea in computer-system design for graphics is that graphical display places too heavy a demand for processing to be handled by the (or a) main, central processor and that there should, therefore, be a satellite graphical processor for each graphics console or cluster of graphics consoles. At present, the satellite processor is usually a small general-purpose computer augmented by a display processor.

The design of the satellite processor is itself an interesting problem. Is it to be programmed once and for all and viewed thereafter as just part of the hardware, or is it to be thought of as remaining a programmable computer? Is its display processor to do nothing but display, or should it be able to call display subroutines and perhaps handle the integer arithmetic of indexing? Sutherland and Myer[11] have observed the tendency for the display processor to evolve into a general purpose computer and then to need a subordinate display processor, and they have determined how and where to stop the potentially infinite regress.

The aspect of system design that I think has not been thought through clearly concerns the division of tasks between the central computer and the satellite computers. There is a watershed: either all the satellites will be near the central computer, or at least one will be remote. If all are near, why should each satellite computer have a separate memory? That leads to a wasteful transferring of data from one memory to another. Why shouldn't the satellite processor be precisely a display processor and address a block of the main memory?

If the satellite is remote, of course it cannot address the main memory directly; it must have a memory of its own. If one satellite is remote, all the satellites should have memories of their own, for it is most important to preserve homogeneity for the sake of simplicity.

Now we come to Bert Sutherland's Dictum: "Think Network!" Even if you plan to have all the graphics consoles in the same room as the main computer, treat them as if remote—because one day soon it will be desirable to admit a remote satellite into the system—or you will want to share software with a system that has remote satellites.

The key problem then becomes the division of tasks and the specification of the interface between the central and the satellite computers. That problem must be solved in such a way that the satellite does not pester the central computer continually, yet the user should feel that he is interacting directly with the central machine. I doubt that there is a good solution.

Economics

There are of course two main kinds of cost: (1) the cost of the equipment; (2) the cost of developing the methods and preparing the programs. The bad thing is that for sophisticated computer graphics both are very great. The good things are that the hardware costs tend to drop rapidly, once they start dropping, and that, as soon as there is replication of hardware, the software costs increase much less rapidly than the number of hardware systems in which the software is used. That is elementary, but it is very important and evidently not well enough recognized.

During the 25-year history of digital computing, the costs of arithmetic units, control units, and processible memories have halved approximately every two years. On-line terminals have not had such a "long" history—at least not in significant quantity—but it appears that some of the simple keyboard-and-cathode-ray-tube consoles that are coming into use in time-sharing systems dropped from about $13,000 to about $6,500 in the year between the last two Fall Joint Computer Conferences. If some are priced at $5,000 in the exhibit area of this conference, it will suggest that, for a time, one class of equipment will be halving in cost each year.

I do not want to make too much of such gross trend analysis, even though it does point to the source of much of the magic of our field, and I certainly shall not base anything on a halving of cost each year. However, the question of optimal research lead time is a very important question, and it needs an answer. When should we develop the kind of computer graphics that will most strongly augment the intellect? When should we start if we want to have it ready when it will be affordable? In their paper, Huggins and Entwisle[7] will call educational application of interactive computer graphics, based on time sharing, an "economic absurdity" and suggest that the facilities be used to generate educational films. Insofar as operations are concerned, I agree with them. But what about research?

As basis for a rough calculation, let me take a system that I think would make an excellent base for a computer graphics laboratory. A few months ago I determined the cost of a graphics-oriented PDP-10 computer with 256K words of 2.5-microsecond memory, 15 million words of disk storage, 16 consoles with storage-tube displays, and quite an assortment of supporting equipment. It was about $700,000. If it were used 8 years, with an average of 10 consoles active over the 24-hour day, the cost per console hour would be $1. Double that to cover maintenance, operation, supplies, and overhead, and you have $2 per console hour to use as an argument that you could be within hailing distance of economic operation now if you had the programs.

As I see it, the advances in methodology and software required to achieve the kind of computer graphics and the kind of interactive dynamic modeling I hope for will require about as much research effort and time as several laboratories have devoted to interactive computing since 1961. If we look ahead to 1977, and assume hard work in the interim, I think we can count on a fairly good methodological and programming capability. If the halving-every-two-years rule held, the over-all equipment cost would be about 12 cents per console hour in 1977. It need not be that low to support all the educational and personal applications that it should support. Other applications—in management, research, medicine, etc.—would have become economic earlier, of course.

My conclusion—which is of course as tentative and open to bias as the foregoing estimates, and perhaps as simpleminded—is that now is the time to push forward with research in the areas of computer graphics that people call expensive, sophisticated, esoteric, and exotic. I think that they are the areas in which lie the real promises of significant improvement in our intellectual processes. If we do not push forward with research in those areas now, we shall find ourselves with a magic lantern that we don't know how to rub.

REFERENCES

1 T STOCKHAM A V OPENHEIM R W SCHAFER
Nonlinear filtering of multiplied and convolved signals
Proceedings of the IEEE 56 August 1968 126–1291

2 J E WARNOCK
A hidden line algorithm for halftone picture representation
Technical Report 4–5 May 1968 University of Utah

3 L G ROBERTS
Homogeneous matrix representation and manipulation of N-dimensional constructs
Lincoln Laboratory Massachusetts Institute of Technology July 1966

4 R F SPROULL I E SUTHERLAND
A clipping divider
AFIPS Fall Joint Computer Conference Proceedings 33–1 December 1968 765–775 (Published separately by the Evans-Sutherland Computer Corporation Research Park Salt Lake City Utah 1968)

5 A C TRAUB
Stereoscopic display using rapid varifocal mirror oscillations
Applied Optics 6 1967 1085–1087

6 J C R LICKLIDER
Computer graphics as a medium of artistic expression
273–304 in Computers and Their Potential Applications In Museums New York Arno Press 1969

7 W H HUGGINS D R ENTWISLE
Computer animation for the academic community
AFIPS Spring Joint Computer Conference 1969

8 T O ELLIS W L SIBLEY
On the development of equitable graphics I/O
IEEE Transactions on Human Factors in Electronics HFE–8 March 1967 15–17

9 D ENGELBART W K ENGLISH
A research center for augmenting human intellect
AFIPS Fall Joint Computer Conference Proceedings 33–1 December 1968 395–410

10 N NEGROPONTE
The architecture machine
Forthcoming The MIT Press 1969

11 I E SUTHERLAND T H MYER
On the design of display processors
Comm of the ACM 11 June 1968 410–414

Computer animation for the academic community

by W.H. HUGGINS and DORIS R. ENTWISLE

The Johns Hopkins University
Baltimore, Maryland

INTRODUCTION

The use of computer-graphic technology to produce low-cost films for education promises enormous educational benefit at modest cost. For educators, these technical developments are but a means to an end which has thus far received too little attention—the production of visual images that in their ability to communicate ideas are superior to traditional graphical images on paper or blackboard.

The printed word is *symbolic*, whereas the TV image is primarily *iconic*. New modes of iconic display may be needed to communicate with young people of the TV generation. Here, computer animation offers remarkable possibilities—instead of static images, words, and mathematical symbols, we may create dynamic signs that move about and develop in self-explanatory ways to express abstract relations and concepts.

Recent research emphasizes the importance of these possibilities. According to Bruner, intellectual development moves from *enactive* through *iconic* to *symbolic* representations of the world, with each level serving to define and give meaning to the next higher level. Education today primarily proceeds at the symbolic level; iconic modes of communication and instruction remain virtually untapped and warrant much more attention. In effecting this, computer animation is certain to play a major role.

The cost of showing

Major efforts to develop films for the classroom have been made during the past decade by various groups interested in improving education in fields such as mathematics, physics, and engineering. Most of these films have attempted to recreate in the classroom experiments and views of real phenomena that the student would otherwise miss. These films are of high quality and made by professionals at places such as the film studios of the Education Development Center, Newton, Mass., in cooperation with one or more principals from the academic community. They have been costly to produce (around $2,000 per minute of finished film). Because of the growing use of these visual materials in schools, however, they can be justified as a worthwhile investment that will reap educational benefits for years to come.

Recent advances in computer-graphic technology are obviously of great interest to educators. Of these graphical techniques, computer-generated film offers many attractive potentialities for the same reasons that studio-made films do. The production of elaborate visual sequences under interactive control of a human makes excellent economic sense for educational purposes provided the final product is recorded on film (or video-tape) for low-cost duplication and distribution to many other viewers. We consider the notion that these sequences should be produced individually under the interactive control of a single student for his sole benefit to be an economic absurdity at present. The realities of the high cost of computing are all too evident to those of us at schools struggling to support even batch-processing utilization of computers for education. Hence, to those wealthy few, who are fortunate to have such graphical displays available, we direct a plea that they should consider arrangements by which other interested members of the academic community can use their facilities for the very beneficial production of computer-animated films for the entire community.

In the remaining portion of this paper, we would like to discuss some aspects of computer animation that have received sparse attention and yet which are vital, we feel, to the ultimate success of films as an educational device. We have heard many papers and much discussion of hardware and software for computer graphics at recent meetings, but virtually nothing concerning the design of appropriate symbols and conventions for portraying ideas and concepts. Enormously elaborate and costly equipment is used to produce images which are primitive and poorly designed to communicate the desired information. The technical apparatus of light-pens and data structures has so monopolized our attention that we have forgotten that all of this is but a means to an end—the production of visual images that are superior and more lucid than the traditional graphical images on paper or blackboard.

Perhaps it is unreasonable to expect those responsible for the development of the hardware and software to also be deeply concerned with the quality and properties of the graphical images that may be produced, but for those of us in education, this latter aspect is of ultimate importance. Some of us are making educational films without benefit of light-pens, online oscilloscopes or immediate access to microfilm plotters.[1,2,3] Typically, these programs have been worked out on paper; the desired sequences translated into some appropriate assembly or compiler language;[4,5,6] the program run on a machine such as an IBM-7094 which creates a magnetic tape that is then sent to an SC-4020 microfilm plotter located in another city where the actual film is produced and returned to the originator after a delay of 3 to 20 days or more! For those accustomed to using interactive graphical displays, such a tortuous process must appear intolerable. Yet, you may be surprised to learn that the long delay has not been as serious as one might expect because the conceptual design of the storyboard and the invention of appropriate conventions and schemes for showing the intended ideas and concepts in an accurate and perceptually clear way is even more time-consuming and difficult than the straightforward task of writing and processing the computer program to make the film.

It is in the art of showing ideas and concepts that educators can make their greatest contribution. Although computer-graphic terminals would be nice to have at every school, these elaborate and costly facilities are by no means essential. For instance, a high-school student, Garrett Jernigan in Raleigh, North Carolina, has written a short animated film to show the earth-moon system in true proportion, and to emphasize that the moon and the earth each revolves around the center of mass of the earth-moon system—a nice lesson in mechanics. The movie starts with a far-out view of our galaxy, then zooms into our solar system. After the earth-moon system is identified as one of the planets moving around the sun, the moon is shown revolving around the earth, with relative sizes and distances accurately portrayed. A further zoom toward the earth reveals on close view that it too is also nutating around the common center of mass. Jernigan didn't have even a computer, but he had the imaginative idea which enabled him to program these sequences in FORTRAN IV punched onto teletype tape. These tapes were mailed to Johns Hopkins where we ran the program on our IBM-7094 and then sent the magnetic tape containing the graphical instructions to Polytechnic Institute of Brooklyn for processing on their SC-4020.

The art of showing

We shall always be grateful to J. C. R. Licklider for bringing to our attention nearly 5 years ago a remarkable book by Gombrich.[7] Until then, we had not been fully aware of how important is cultural conditioning in altering the perception of visual images. Gombrich examines in depth the notion that all art (and visual communication, generally) involves illusion and he shows that a "realistic" representation always incorporates unrealistic conventions that must first be learned and then ignored. In our own culture, where photographic-like representations have been so highly developed, most people would regard a photographic portrait of a human head in profile as a more realistic representation than a Picasso drawing showing *both* eyes in a *profile* view. Yet, if you were to show the two representations to a visually illiterate aborigine, he might select the Picasso as the more realistic because the photograph shows only *one* eye, whereas most people have *two* eyes (as realistically represented by Picasso). The conventions of perspective, which seem so natural and absolute to us, are quite unnatural to the savage who may, in fact, not even recognize a photograph as a picture but see it simply as a blotchy, discolored piece of paper (which it is!). (Incidentally, although visually illiterate people may not perceive the content of a *still* photograph, they always perceive the moving images in a motion picture.[8] This observation emphasizes the likely advantage of computer-animated presentations, particularly for elementary and secondary-school education.)

Unlike the aborigine, nearly all school children in this country today are extremely literate visually. It is estimated that on the average they will have spent between 10,000 to 15,000 hours watching TV by the time they finish school. The impact of TV on our culture today is hard to evaluate, but there is little doubt that it is enormous. For instance, one of us has found[9] that disadvantaged first-graders of the Baltimore inner-city schools have better developed abilities to associate and use common words than the more privileged children of corresponding age in suburban schools. Furthermore, by the time children finish elementary school their verbal associative structures appear to be much further developed than students of the same age who lived 50 years ago or than students who today are members of cultural groups in which TV is little used.[10] That these effects are a direct consequence of television seems very likely; slum children spend much more time watching TV than children in middle-class homes where a larger part of their time is directed by parents and teachers toward other activities.

These young people of the TV generation seem to have developed a different kind of visual literacy than those of us who grew up prior to TV, and new modes of visual display may be needed to communicate effectively with them. The printed word is *symbolic*, whereas the TV image is primarily *iconic*. Yet our traditional modes of communication in science and engineering are still dominated by symbols rather than by the icons to which the TV generation is habituated. Here, computer animation offers remarkable possibilities that have never before existed—instead of static images and mathematical equations on the printed page, we may now create dynamic signs that move about and develop in self-explanatory ways to express abstract relations and concepts. These potentialities for iconic communication of the quantitative ideas central to science and engineering are only now beginning to be exploited; most of the imagery that we have seen produced by computer graphics in CAI and other applications has merely transferred the traditional static signs and symbols from the printed page to the cathode-ray tube. A dynamic dimension is now available that requires the invention and development of new conventions and a visual syntax appropriate to this new medium if it is to be fully used for communication and education. (These possibilities are suggested by the computer pantomimes[1,2] which communicate quantitative concepts without using words or equations.)

The technical advantages of computer-animated films have been discussed elsewhere.[11] Not only is animation produced in this way likely to be much less costly than traditional animation, but one person can do the whole thing, from conception of the idea through programming and production of the final film. It thus becomes feasible to study alternate schemes for displaying certain ideas by developing at low cost a wide range of visual materials useful for empirical tests with student subjects for evaluating the effectiveness of these different schemes.

In making a computer-animated film, if one deliberately avoids the use of traditional words and mathematical symbols and attempts instead to portray all abstract ideas iconically, he quickly learns that "one word is worth a thousand pictures!" He also soon discovers that many of the signs traditionally used in science and engineering are inadequate for conveying many familiar concepts without first introducing irrelevant details freighted with erroneous artifacts and implications. For instance, in some recent research on the design of symbols for representing electric circuits,[6] we found it very difficult to indicate cause-effect relations. (We finally used an inelegant anthropomorphic symbol shaped like a human hand to change the value of the source signal.)

Another unsolved problem is how to portray a continuum field, such as the electric potential around a set of charges. The portrayal of a vector field using the familiar stream lines (or lines of force) leaves much to be desired because the direction of the lines is not easily shown, (e.g., arrowheads introduce broken-line segments which violate the portrayal of smooth continuity of the field). Furthermore, when one attempts to superimpose two such fields, as in demonstrating the superposition of forward- and backward-traveling waves on a transmission line, a whole host of spurious effects result as field lines cross each other or vanish and reappear from nowhere, and perform other atrocities. These difficulties arise even in showing simple, two-dimensional fields; traditional conventions are completely inadequate for portraying general vector fields in three dimensions. New ideas are badly needed for representational schemes that will be free of these bothersome artifacts.

The theory of showing

In his recent important book, Bruner (who is the country's leading cognitive theorist) has suggested[12] that the acquisition and understanding of information generally proceeds through three stages:

1. Manipulative	—personal action
2. Iconic	—perceptual organization and imagery.
3. Symbolic	—use of public language and other private representations.

These stages occur in the development of every child. In the beginning, the child manipulates and experiences things surrounding him in a most intimate and direct way. Later, he recognizes things by their appearance, and the images in his environment acquire an autonomous status as he explores the similarities and differences among the concrete objects around him. The child begins to observe that things are related to other things in more or less predictable ways. As he becomes aware of geometrical and other invariants in his environment, he is able to make predictions and extrapolations from what he perceives on any single occasion and to further refine his internalized model of the world. A major advance occurs when he gives symbolic names to these perceptions and relationships and gradually begins to use words to stand for objects not present. Finally, through absorbing a generalized syntax and semantics, he learns to use words and other symbols to deal with ideas and thoughts for which there are no direct referents in immediate experience.

As intellectual development moves from *enactive* through *iconic* to *symbolic* representations of the world, each level serves to *define* the elements of the next higher level. Like nested macro instructions in a compiler, the abstract symbols expand into more primitive instructions that are often iconic so that the abstract symbols will have concrete meaning. If the learner has a well-defined symbolic system, it may be possible to bypass the first two stages and communicate with him purely at the symbolic level. But too often, the learner may not possess the imagery to fall back on when his symbolic transformations fail to solve the problem, and for many persons it may be impossible to rely entirely on a completely symbolic mode.

We wish to draw an analogy between the process of programming a computer and the process of instructing a student that may (like most analogies) be of questionable value, but it will serve to emphasize a worthy point. A source program in a high-level language, like FORTRAN, is purely symbolic and, by itself, has no meaning to the computer until expanded into a more primitive set of assembly instructions which are recognizable by the machine. Insofar as the computer is concerned, these assembly instructions are iconic. But they, in turn, must be defined in terms of actual manipulative actions performed by the built-in operations of the machine.

In terms of this analogy, the major task of instruction is to provide the computer with a useful compiler, rather than with a multitude of FORTRAN source decks. Certainly no one would attempt to read a FORTRAN program into the computer before a working FORTRAN compiler had been entered into its memory. Yet, the analog of this is attempted every day in the instruction of students. No, that is not quite correct: a FORTRAN compiler is, in fact, first entered—but it, too, is written in FORTRAN!

In his closing paragraph of Chapter 3, (page 72), Bruner states[12]

"Finally, a theory of instruction seeks to take account of the fact that a curriculum reflects not only the nature of knowledge itself but also the nature of the knower and of the knowledge-getting process. It is the enterprise par excellence where the line between subject matter and method grows necessarily indistinct. A body of knowledge, enshrined in a university faculty and embodied in a series of authoritative volumes, is the result of much prior intellectual activity. To instruct someone in these disciplines is not a matter of getting him to commit results to mind. Rather, it is to teach him to participate in the process that makes possible the establishment of knowledge. We teach a subject not to produce little living libraries on that subject, but rather to get a student to think mathematically for himself, to consider matters as an historian does, to take part in the process of knowledge-getting. Knowing is a process, not a product."

CONCLUSION

What, then, may be concluded from all of this? We believe that there is tremendous untapped potential in the use of iconic modes of communication to give fuller definition to the symbols that so dominate the classroom today. There is a paucity of experimental evidence on this issue. Nevertheless, we have prepared a self-instructing text for use at the early college level to teach the major concepts of modern system theory by using the highly iconic notation of signal-flow graphs.[13] Judging from student response this approach has been very effective. In another vein, the studies of Rohwer[14] show that pictorial presentation of pairs of objects to elementary-school children leads to better association between the objects than verbal presentation of pairs of words representing the objects. At the college level, this difference vanishes, reflecting probably the greater symbolic competence of the older person.

What may be true is that the iconic stage, largely ignored, has had an aborted development. Who knows what the potentialities may be? Young children, adults, people who have never been "hooked" on reading, are those most addicted to TV—it is as if there is a vast starvation for meaningful communication that has never been met by the standard media using printing. Instead of bemoaning this, one could exploit it. What would happen, for instance, if even modest sums from Headstart were diverted to producing TV programs of educational use to four-year-olds?

We agree with Bruner when he states (page 34) "that principal emphasis in education should be placed upon skills—skills in handling, in seeing and imaging, and in symbolic operations, particularly as these relate to the technologies that have made them so powerful in their human expression." He mentions the increased visual power and subtlety of students exposed to courses in visual design, and the experiments by Holton and Purcell at Harvard with instruction in visual patterns as a mode of increasing the ability of physics students to represent events visually and non-metrically. He believes "that we have not begun to scratch the surface of training in visualization—whether related to the arts, to science, or simply to the pleasures of viewing our environment more richly."

We believe these theoretical considerations justify much more effort toward the development of iconic modes of communication, and that computer animation can play a major role in these developments, both at the research level and in the classroom. In particular, the subject matter of system theory offers many interesting opportunities for visualizing the topological and dynamic relationships that occur in models of many fields. It is a stimulating and challenging area for study. We commend it to you.

REFERENCES

1 W H HUGGINS D WEINER
Harmonic phasors
7-minute, 16 mm black and white silent (computer pantomime) film

2 D WEINER W H HUGGINS
Response of a resonant system to a frequency step
12-minute, 16 mm black and white, silent (computer pantomime) film

3 J R MELCHER
Complex waves I: Propagation, evanescence, and instability
26-minute, 16 mm black and white, sound film
Complex waves II: Instability, convection and amplification
23-minute, 16 mm black and white, sound film
Note: All of these films were made under the auspices of the National Committee on Electrical Engineering Films and are obtainable on loan from Education Development Center, 39 Chapel Street, Newton, Massachusetts 02160.

4 K C KNOWLTON
A computer technique for the production of animated movies
Proc AFIPS Conference 1964

5 F J SARNO
Polygraphics users manual for the SC-4020
Polytechnic Institute of Brooklyn 333 Jay Street Brooklyn New York 11201

6 W H HUGGINS D ENTWISLE
Computer animated films for engineering education
Final report under Grant OEG2-7-062816-3097 U S Office of Education September 30 1968

7 E H J GOMBRICH
Art and illusion
Bollingen Series XXXV 5 Pantheon Books 1961

8 M H SEGAL D T CAMPBELL
M J HERSKOVITS
The influence of culture on visual perception
Bobbs-Merill Co Inc Indianapolis 1966

9 D R ENTWISLE
Developmental sociolinguistics: Inner city children
American Jour of Sociology Vol 74 1968 37-49

10 D R ENTWISLE
Developmental Sociolinguistics: A comparative study in four subcultural settings
Sociometry Vol 29 1966 67-84

11 K C KNOWLTON
Computer-produced movies
Science Vol 150 November 26 1965 1116-1120

12 J S BRUNER
Toward a theory of instruction
Harvary University Press Cambridge Mass 1967.
See especially Chapters 1 and 2

13 W H HUGGINS D R ENTWISLE
Introductory systems and design
Ginn/Blaisdell Pub Co Waltham Mass 1968

14 W D ROHWER
Social class differences in the role of linguistic structures in paired-associate learning
Final Report—Project No 5-0605 Office of Education November 1967

Graphics in time-sharing: A summary of the TX-2 experience*

by WILLIAM R. SUTHERLAND and JAMES W. FORGIE
Massachusetts Institute of Technology
Lexington, Massachusetts

and

MARIE V. MORELLO
Keydata Associates
Watertown, Massachusetts

INTRODUCTION

The TX-2 computer, an experimental machine at the M.I.T. Lincoln Laboratory, has been in operation for almost 10 years as an on-line, graphically oriented facility.[1] In 1964, a time-sharing system for the TX–2 was started. This system, APEX,[2] was to service a small number of consoles with graphic display capability. To achieve hardware economy, displays were to be refreshed from main core memory through a time-shared vector generator providing analog signals distributed to the individual console scopes. The displays were to be refreshed directly from a structured display file as experience with the Sketchpad developments[3,4] of the early 1960's had indicated was highly desirable for interactive graphic applications. Although the APEX graphic system has evolved through several generations of display hardware and corresponding software changes, the initial design principles of displays refreshed from structured information in main core by a time-shared generator have remained. This paper is an attempt to collect and evaluate some lessons learned from our experience in developing and using this system.

TX–2 is a 36-bit, fixed-point machine currently with 164,864 words of storage; 139,264 words of 2 μsec or faster core and 25,600 of 1 μsec thin film.[5] As part of the APEX system development, rather elaborate memory address mapping hardware was added to the computer.[2] A Fastrand II drum provides both bulk and swapping storage for the system. The TX–2 operation under APEX differs from many contemporary time-sharing systems in that most (6 of 9) of the user consoles are physically located in the computer room and are equipped with display hardware. Limited remote access is possible via teletype, and a single remote display console can be used on the system.

In the now customary fashion, the APEX executive system provides a virtual computer for each console. For reasons of resource allocation, scheduling, and protection, input and output operations are not performed directly by user programs but are handled as executive services. The APEX system provides for CRT output of graphical information at each console, and gathers and buffers interactive inputs. These features are invoked from user programs by supervisor calls on the APEX system.

Display output

All local display scopes are driven from a single analog signal generator which handles points, vectors, curves, and characters.[6,7,8] The output of this generator is time multiplexed on a frame-to-frame basis with individual intensification signals determining which scope (or scopes) will display a particular frame. Both refreshed and storage displays are available as well as a drysilver printer which can be switched manually to obtain a hard copy of any console's display. The speed of the display generator output is controlled by the display monitor routines to compensate for the differences in useful writing rates between

* This work was supported in part by the U.S. Air Force and under a subcontract of M.I.T. Lincoln Laboratory, with support from the U.S. Advanced Research Projects Agency.

refreshed and storage scopes. The system currently handles five refreshed displays which appear to be about the maximum that can be handled with acceptable flicker under our normal use conditions. There is no such limitation on the number of storage displays which could be handled, but in a mixed system the painting of a complex picture on a storage scope has a serious effect on the refreshed scopes because of the slow writing rate in the storage unit. Some consoles have both refreshed and storage scopes.

The display generator is driven by a channel from data buffered in main core. The channel as well as the main processor uses the address transformation hardware of TX-2. As a result, a buffer file of display data need not occupy physically contiguous core pages and can be handled by the normal dynamic core allocation mechanism of APEX. The buffer pages are always in core so that a user's display may be refreshed even when his program and problem data have been swapped out to the drum. The display monitor routines are resident in core only when display services are in use.

Displays at any console are controlled by supervisor calls from the user program associated with that console. APEX builds and maintains structured display files for the refreshed scopes, but simply transmits a linear buffer of data to the storage scopes. In the latter case the buffer remains accessible to the user program except when the display is actually being painted. The display information provided by the user program is in a form directly usable by the display generator hardware. This hardware-oriented format permits direct program manipulation of low-level display data with resulting good performance but causes changes in the hardware to reflect back into user programs.

For the refreshed and structured display the supervisor calls are of four basic kinds: on-off, item, group, and report. The on-off call requests or releases display services at a console and initializes a user's display file buffer. The report call allows a program to interrogate an existing display file to discover its hierarchical structure. This call is useful when working with a display buffer built by another program.

The item and group calls reflect the conceptual hierarchy of the display structure. An item is a collection of points, lines, curves, or characters with a single identifying name; it is the smallest display unit, but may contain many display components and create a complex picture. A group is an identified collection of items and may also contain "uses" of other groups. A use is an offset reference to another group which then serves as a sub-picture. Only translation of sub-picture references is possible; the display hardware does not permit dynamic rotation, scaling, or scissoring.

The display hierarchy starts at a reserved group with identifying name 0 (zero), and anything contained in its substructure will appear on the scope. Groups which are not subordinate at some level to group zero are invisible. Thus, at a cost in frozen core requirements, two versions of a picture can be built in separate groups and kept in the display file for rapid switching of actual display between them.

The display file maintained by APEX for each console has user-supplied identifying names associated with each item and group. These names allow for future references to the display components. For example, a call to insert a new item into a particular group does an automatic replace if an item with the same identifying name already exists in the group. The identifying names are also returned by input monitor routines, when hits on the display are recorded.

For structured display files, dynamic storage allocation, garbage collection, and ring structuring are handled automatically by APEX with no attention required from the user. The display file for each console is initially three pages (each 256 words) long and is expanded when needed up to thirty-two pages. Since the display file is always in core, expansion is done only when necessary and then only to the amount needed. While the three page minimum is adequate for typical editing jobs, the thirty-two page maximum has occasionally been an annoying limitation for large complex pictures.

Console input

The input hardware attached to TX-2 consoles has included light pens, RAND and Sylvania tablets,[9,10] shaft encoder knobs, switches, and pushbuttons. Not all consoles have the same configuration, but a desirable choice has emerged. Light pens have been abandoned in favor of tablets for graphical input. A hardware comparator[11] provides a "light-pen-like" interrupt when the analog deflection signal passes close enough to the tablet's pen position. A small number of buttons or switches is also useful as is the standard alphanumeric keyboard. Knobs are available on some consoles.

The monitoring of all on-line console input devices is handled by input monitor routines in the APEX system. These allow user programs to request and release input services and to specify the conditions under which the executive should interrupt programs to provide the requested information. Information from the requested inputs is stored for the user program in an accessible buffer.

The input buffer is another stay-in-core file, enabling the executive to accept inputs from the user's console even when his program is inactive on the drum. The circular buffer is 256 registers long and contains a list of up to 75 independent entries. Each entry contains information identifying the device which caused it, a mask stating which associated devices are being reported in this entry, and the input values from those associated devices.

For example, a program might request an input entry whenever a light button was hit (with tablet and comparator) and with that event a report of the value of the real time clock and the value of the knob register. Another request might specify a button push as the event and request an associated report of toggle register values. In addition to console buttons any desired set of keys on the standard alphanumeric keyboard may be used as control buttons with the remaining keys serving in the normal fashion.

Monitor routines, when requested, automatically provide for the display of tablet input data. A tracking dot on the screen follows the stylus motions, and an "ink" trail is generated when stylus strokes are drawn. Tablet track data is available to user-programs in a separate extension of the input buffer. Only a limited number of tablet points (256) are buffered after which the pen runs out of ink. For most tablet usage, this restriction has been reasonable, but it can upon occasion prove extremely aggravating. The track input data is normally used by the public symbol recognition system described later.

338 remote display console

A Digital Equipment Corporation 338 display console with 16K of core has been attached to TX-2. The console operates remotely in Washington, D.C. via a leased 1200 bit/sec line. This remote console was intended to be compatible with existing programs previously developed for local console operation, but this goal has been met only partially. Typed communication can be handled both to and from the console, and display output can be sent to the 338, but input from devices other than the keyboard of the remote console is not allowed, and the display report call has not been implemented. Because the remote console is equipped with a standard TX-2 keyboard and its output display has been coded to handle the special TX-2 character set, many standard TX-2 programs are usable from the remote terminal.

To operate this remote display, APEX provides special treatment for calls from programs being run remotely. Typed output data is first converted from the TX-2's character set and codes into an ASCII format and then sent to the 338 which converts again to its internal format. Typed input is handled similarly, with transmission in ASCII and conversion at both ends. Since display output data is passed from programs to the executive in a format determined by the TX-2 display generator hardware, and it is obviously undesirable for a remote user to be concerned with this format, the data is reformatted into a hardware independent picture description and sent as a binary message to the 338. The 338 then converts the binary picture description to a format suitable to its display hardware. Interactive input from the 338, if implemented, would be handled as binary data to be converted into standard TX-2 input buffer formats.

With the exception of handling communication protocol, the 338 is functionally similar to a local console, and this remote console computer is not used in any complex way as is done elsewhere, e.g., Graphic I at Bell Telephone Lab.[12] The goal was to run remotely graphical programs developed locally at TX-2 without requiring program changes.

Graphics applications

One early and straightforward use of displays was in the Reckoner system.[13] This system allows an engineer unfamiliar with programming to sit at a console and easily manipulate arrays of data. He operates one step at a time from a menu of numerical operations. Graphs plotted on the display scope are a common form of output. Service programs for this plotting provide automatic scaling of axis markings and are invoked simply by the Reckoner user.

Because many users of the TX-2 facility are system creators, the most common use of displays is in program editing and debugging. The TX-2 assembler and compilers all link to a standard text editor which allows changes to be made on-line to a page of program source text displayed on the screen. Special keys (part of the console keyboard) position a cursor in the text for controlling insertions and deletions. The editor is available as an easily called service, and is used by many sub-system programs. In particular, this editor is available from the control section of compilers produced by the TX-2 compiler-compiler system.[14] When a source error is found during a compilation, a single-key edit command will automatically display the bad source line and its neighbors for immediate editing. Duing run-time debugging, displays are used to show data values, machine state, and other pertinent information. Control commands can be given on-line from the keyboard or can be automatically retrieved from a previously saved text file.

One novel application of displays deals with the problems of debugging programs which use the display services.[15] What does a user do when a program runs, but the picture he expects to see on the screen is either missing or incorrect? Core dumps of the display data maintained by the executive are difficult to interpret. A public program is available to interrogate a display file maintained by the executive and to present a picture of its item and group hierarchy on the display. A typical discovery that a user might then make is that his program did indeed make a correct display structure, but since he erroneously calculated zero length for all lines, the display was properly blank. This operation need not disturb the user program which can resume with its own display restored after this analysis has been viewed.

In order to facilitate the use of tablet input in user programs a public symbol recognition system has recently been created.[16] It consists of a basic recognition program controlled by individual feature dictionaries. A separate trainer program builds these feature dictionaries from sample symbols drawn on the console tablet. Each user can have his own individual style and indeed can have several dictionaries for different applications and change them at will. When invoking the recognition service, the name of the symbol dictionary to be used is passed as a parameter. Standard recognition techniques are used[17] and the recognizer is not very sophisticated. In particular, the recognition scheme is stroke oriented and uses a grid of overlapping zones superimposed on each stroke. The recognizer returns a code for the drawn symbol as well as position and size information. This recognition system has been used in a number of applications including a tablet editor and programs for circuit design. Since APEX provides a push-down stack of virtual memories[2] for the user's pseudo-computer, it is very simple to retrain and revise the feature dictionary in the middle of a job in case of trouble with a certain few symbols. The speed and accuracy of the recognizer are adequate for on-line program control applications, using up to a few dozen symbols.

High-level methods for expressing scope output and console input operations have produced a great deal of display programming activity. The obvious advantages over assembly coding of clarity, brevity, and fewer mistakes are a strong incentive for users. The compiler-compiler system on TX-2 has made relatively easy the implementation and evolution of an extended high-level language based on ALGOL. The language, called LEAP (**L**anguage for **E**xpressing **A**ssociative **P**rocedures)[18] has associative data structuring operations, reserved procedure forms for display or input manipulation, and real time variables such as the clock time and tablet stylus coordinates. Direct means for invoking the symbol recognizer's services are even incorporated. A "Recognize" statement gets a symbol from the tablet just as a "Read" statement gets a symbol from the console keyboard. Writing interactive programs which use the display is straightforward, and experimentation and modification can be rapid.

Having LEAP available as a programming tool has facilitated the evolutionary development of application programs for graphical programming[19], data analysis, logic diagram input[20], and integrated circuit mask layout. The largest effort has been on circuit mask programs.[21,22], A circuit designer controls the mask layout program with freehand figures sketched on the Sylvania tablet. The computer recognizes his rough marks as commands to create, move, group, and delete various integrated circuit components. Once a circuit design is complete, output tapes for each of the mask levels required can be punched for later use by a precision pattern making machine. Individual variations among designers in drawing style are accommodated easily by the trainable recognizer.

Another application has been an experimental animation system.[23] The animator first sketches the basic picture parts which will appear in the film frames. He then sketches pictures of a different kind; pictures which define the dynamics of the film and which determine how the basic frame components will move and appear in the film. This animation system is an example of an application where the picture information can easily exceed the capacity of core memory and where the computing needed to calculate the picture from a more compact description exceeds real-time capabilities. To meet this kind of requirement, APEX allows the structured display files it has built to be saved in bulk storage as ordinary user files. When a user program presents such a file to the executive for redisplay, a quick check is made of the file contents, and if the check is passed, the file is immediately displayed. If moderate to large size display files are used for such a purpose, they can be retrieved from bulk storage in considerably less time than is required to paint the display they represent, and a long, essentially continuous chain of pictures can be presented. Of course, in this sort of application the presence of other display users in the time-sharing system will disturb the continuity of the picture sequence.

It will be noted that most of the graphics application discussed here have been in sub-systems created for a variety of purposes within the APEX environment.

By using the graphic capability in unforeseen as well as obvious ways the creators of these sub-systems have made the use of on-line displays the accepted operating norm.

Critique

Success in interactive computer graphics requires first that a programmer have access to convenient programming tools for making interactive graphical programs, and second that these programs run well in their operating environment.

In the first of these areas, the programming tools available on TX–2 are a mixed lot. At the level of assembly code, the supervisor calls necessary to use the graphic capabilities lack a convenient symbolic notation. The high-level display and input constructs available in LEAP have been a success largely because the details of the supervisor calls are hidden from the programmer behind a consistent symbolism. The well-known lesson relearned again is that notation is very important, and that a clear symbolic means for programming display and input actions is necessary. The flexibility of a compiler-compiler system has been important in the trial and error development of high-level programming tools for graphics. The LEAP compiler has been modified continually as programmers discovered useful constructs for interactive programming. As a result of this unstructured growth, however, LEAP is not a formally neat language, and its run-time routines impose a large load on the time-sharing system.

The particular display structure concepts implemented in APEX have been generally satisfactory. For some applications (the text editor for example) the structure is unnecessarily complex, and access to it is too restricted. A simple linear buffer shared by user and supervisor can lead to a more efficient design in such an application. This scheme, which allows the user program to structure the buffer space to suit its requirements, works well for storage displays but poses a problem for refreshed displays. If the user program is allowed to modify the display information while the display is actually being painted, undesirable artifacts are likely to appear on the scope. The simple expedient of stopping the refreshing action while making the changes will produce serious interruptions in display continuity in the event the time-sharing scheduler switches users while the change is taking place. In the APEX system these problems are overcome by having the supervisor build and maintain the structured display files, and considerable complexity in both the structures and the supervisor routines has resulted from the requirement of maintaining display continuity while changes were being effected. Our conclusion here is that while highly structured displays supported by a complex supervisor system are well suited to many applications involving highly dynamic interactive graphics, simple unstructured displays are adequate for much of the work that a system such as APEX supports, and a well balanced system should provide both services.

The criterion that interactive graphical programs run well in their operating environment presents many problems. The APEX scheduler is designed to provide very fast response to user interactions by giving control to a user program within milliseconds following an interrupt regardless of the program's present position in the scheduling queue. However, if the responding program and the data files it requires are not in core memory, swapping must occur, and with current secondary memory hardware this means that the user program will not actually begin to run for a time varying from several hundred milliseconds to several seconds depending on the size of swap required. An obvious but expensive solution for us is to add faster secondary memory hardware to the system. In the absence of that hardware, the only remaining course of action is to keep the responding programs and data in memory. APEX allows a user program to specify that as much as two thousand words of program and/or data be kept frozen in core memory. By judicious use of this capability the scope editor achieves very good response even under extreme time-sharing load conditions. Unfortunately, many interactive programs, particularly those built using the higher level language facilities, are too large to be operated in frozen core. Their users either struggle along with inadequate response, or if the application is sufficiently important, achieve core residency by administratively eliminating other usage of the time-sharing system while they operate. Our conclusion here is that further work on the higher level languages is needed, so that the conceptual advantages they offer can be obtained with manageable core load and computer time requirements. Since it is unlikely that the overall run-time programs could be kept sufficiently small to remain frozen in core memory, it becomes desirable to give the end-use programmer (as opposed to the compiler writer) more control over the run-time system. He might then be able to partition his run-time program to take advantage of the frozen core capability.

The performance of a graphics system depends heavily on the performance of the display hardware. As one might expect, running five displays from one generator results in heavy flicker in the worst cases.

The display generator itself has been modified several times and replaced once to obtain speed improvements. In the present configuration performance is limited by the deflection circuitry in the display scopes themselves. Good quality lines are drawn at a rate of 25 μsec per inch. Character generator speed of 80 μsec per character is set for storage display use since the character generator speed is not under program control. Refresh rates slower than one frame per second per display have been encountered. A significant benefit has been obtained by using a P12 phosphor instead of the more usual P7. Without light pens, the need for the P7's bright blue flash has been eliminated, and the yellow P12 provides a much more restful picture. One conclusion is that in many cases a badly flickering display is more useful than none at all. For small text editing jobs in particular, the scopes are used universally in preference to a keyboard editor no matter how bad the flicker. The flckier experienced by a user with a modest picture could be improved at the expense of a user with a large complex picture if the display generator could be shared on an equal-time rather than an equal-number-of-frames basis. Equal-time sharing of the generator requires that its internal state be accessible to the supervisor program, a feature lacking in the present hardware.

Buffering the five displays out of main core and having the main frame handle all console input interrupts exacts a penalty in overhead. Measurements show that display refreshing can consume up to 15 percent of the available memory cycles, although typical displays require well under 10 percent. The overhead depends heavily upon the use of structuring in the displays. Because the channel hardware is not capable of handling the closed sub-routine structure used in some applications, main frame attention and extra core cycles are required to effect subroutining. Channel hardware to perform this task is now commercially available. The use of a single time-shared generator makes the refreshing overhead independent of the number of displays being refreshed. There is no such advantage working in favor of tables each of which requires main frame attention at 5 millisecond intervals. Tracking of tablet position with a moving dot on the scope costs 5 percent in overhead for each tablet. Inking, the display and buffering of a stylus track, boosts the overhead momentarily. Clearly, if many tablets were to be supported, the system would require some peripheral processing to keep the overhead within reason. Display and tablet overhead have little effect on response time, which is dominated by swapping considerations, but the slowdown from memory interference can be seen in its effect on large computation jobs.

A controversial aspect of the TX-2 graphics system design is its use of a hardware oriented format for display information. Besides the obvious disadvantage of causing hardware changes to reflect back into user programs when they might otherwise be absorbed in supervisor code, this hardware dependence causes a conceptually unnecessary translation to be made when a remote display or another computer in a network is the ultimate receiver of the display information. If the display format were restricted to a hardware independent format such as is used by a high level language like LEAP, it might be possible to avoid any translation for remote transmission. An advantage of a hardware-oriented format in an experimental system like TX-2 is that the user programmer can immediately make use of any new hardware features as soon as they are made available. If he is restricted to a hardware-independent format he will have to wait for some system programmer to complete the necessary translation changes, or worse, wait for some committee to decide on how the existing format is to be extended to encompass the new feature. Our conclusion is that systems intended to support developmental work in computer graphics should provide for both kinds of formats with translation as necessary. We would like to think that it may one day be feasible for hardware to be designed which would directly accept display information in an optimum hardware-independent format.

Our experience with the 338 remote display console indicates a problem of major importance in future system design. Our choice of interfacing the remote console at the level of APEX supervisor calls had the advantage of limiting responsibility to a few people concerned with the supervisor program. It was unnecessary for the writers of sub-system programs such as editors and compilers to be concerned with the nature of the console they were serving. This choice made major portions of the TX-2 software system available to the remote console. Unfortunately, many of these programs do not run well on the remote console, indicating a mismatch between system interface and communications requirements. For example, the text editor replaces the entire visible page of text in the display structure whenever the cursor is moved or a change is made. This action is almost instantaneous on a local console but may take several seconds on the remote console because of the time required to transmit the new page over the phone line. Clearly, there are many solutions to this particular problem, but they all involve taking into account the problems of remote usage at an earlier point in the design of basic sub-system services. The entire editing job could be transferred to the remote station if an appropriate partitioning place in the overall system could be identified. For networks of time-shared computers to flourish, programs must be written from inception with network partitions in mind even though the initial usage will be local. The programming tools available must encompass network concepts so that new programs will be network compatible as a matter of course. The alternative of patching programs after the fact for network usage is inefficient, and considering human nature, probably unworkable.

CONCLUSIONS

The time-shared use of displays in APEX has been satisfactory in many respects. Much productive work has been assisted by the use of the scope text editor and by the graphic display of data. The capability for experimentation with graphical programs provided by the high-level programming tools has permitted the rapid evolution of application programs.

Creating useful interactive graphical programs can be a long, difficult task. One difficulty is that the end user often does not understand his application area in sufficient detail to predetermine a satisfactory mode of console communication to and from the computer. Trial and error development may be the only feasible course, providing that the application programs can be created and changed easily. The TX-2 environment has provided a significant capability for interactive program development and experimentation.

The most critical problem area remaining in general is that of program partitioning. For reasons of response, economics, and networking among others, a trend toward multi-computer application programs is discernible. These large programs with many pieces must be created and modified with reasonable effort, and documented interface information must be generated as a byproduct of program development. Programming language constructs and tools for dealing with multi-computer environments are needed. Operating systems must incorporate appropriate I/O concepts and permit conventions for easy linkage to programs operating in remote and different computers.

ACKNOWLEDGMENT

The authors wish to recognize the contributions of the many people who have participated in developing and using the APEX system on TX-2.

REFERENCES

1 W A CLARK et al
The Lincoln TX-2 computer
Proc Western J C C February 1957

2 J W FORGIE
A time- and memory-sharing executive program for quick-response on-line applications
Proc F J C C 1965

3 I E SUTHERLAND
SKETCHPAD: a man-machine graphical communication system
Proc S J C C 1963

4 T E JOHNSON
SKETCHPAD III: a computer program for drawing in three dimensions
Proc S J C C 1963

5 J I RAFFEL et al
A progress report on large capacity magnetic film memory development
Proc S J C C 1968

6 T E JOHNSON
Analog generator for real-time display of curves
M I T Lincoln Laboratory Technical Report No 398
July 1965

7 L G ROBERTS
Conic display generator using multiplying digital-analog converters
IEEE Trans on Elec Computers EC-16 June 1967

8 H BLATT
Conic display generator using multiplying digital-analog decoders
Proc F J C C 1967

9 M R DAVIS T O ELLIS
The RAND tablet: a man-machine communication device
Proc F J C C 1964

10 J F TEIXERA R P SALLEN
The Sylvania data tablet
Proc S J C C 1968

11 K H KONKLE
An analog comparator as a pseudo-light pen for computer displays
T IEEE on Computers C-17 January 1968

12 W H NINKE
Graphic 1—a remote graphical display console system
Proc F J C C 1965

13 A N STOWE R A WIESEN D B YNTEMA J W FORGIE
The Lincoln reckoner: an operation-oriented on-line facility with distributed control
Proc F J C C 1966

14 L F MONDSHEIN
VITAL compiler-compiler system reference manual
M I T Lincoln Laboratory Technical Note 1967-12
February 12 1967

15 R M BAECKER
Experiments in on-line graphical debugging: the interrogation of complex data structures
Proc International Conference on System Sciences
January 1968

16 J E CURRY
A tablet input facility for an interactive graphics system
International Conference on Artificial Intelligence
May 1969 (submitted)

17 W TEITELMAN
Real-time recognition of hand-drawn characters
Proc F J C C 1964

18 P D ROVNER J A FELDMAN
The LEAP language and data structure
IFIP Congress 1968 August 1968

19 P D ROVNER D A HENDERSON JR
On the implementation of AMBIT/G: a graphical programming language
International Conference on Artificial Intelligence May 1969 (submitted)

20 G D HORNBUCKLE R N SPANN
Diagnosis of single gate failures in combinational circuits
International Solid State Circuits Conference February 1969

21 W R SUTHERLAND
Computer assistance in the layout of integrated circuit masks
IEEE International Convention Digest 1968

22 D J ECKL K H KONKLE W R SUTHERLAND S A IDZIK R L LUCE
A subnanosecond ECL circuit produced with the aid of computer graphics
IEEE International Electron Devices Meeting October 1968

23 R M BAECKER
Picture-driven animation
Proc S J C C 1969

Teaching heart function—One application of medical computer animation

by ALLAN H. GOTT and BRUCE R. KUBERT

The Aerospace Corporation
San Bernardino, California

and

ALLEN F. BOWYER and GEORGE W. NEVATT

Loma Linda University
Loma Linda, California

INTRODUCTION

Medicine has developed current methods of teaching heart function over several centuries. There have been few fundamental changes in these teaching techniques over the last several decades. The method of text study followed by laboratory activity is essentially the same for both undergraduate and medical school students. For the latter, a more comprehensive study of experimental animals is involved, and surgical procedures are observed on patients. Once in medical school, almost all teaching is by demonstration, with a very small number of students per instructor.

This technique of demonstration and individual instruction limits the number of detailed phenomena which may be taught simultaneously, and the rate at which they may be presented. However, the complete description of heart function includes information of several types. The volume, configuration, and movement of chambers, valve operation, pressure and flow, electrical waveforms, and heart sounds all yield meaningful information. Specialized instruments provide the cardiophysiologist with massive amounts of raw data relating to each characteristic. Such data are often subjective, and when many different types are brought together, are usually unintelligible to medical students, and in fact to practicing physicians without special training. Thus, a major task of heart researchers and medical educators is to devise methods to efficiently integrate and disseminate heart function information. As the volume of heart physiology and research data increases, there is a mounting need to present these complex interrelationships clearly and simply. This paper is concerned with the application of computer techniques to teaching heart function.

The short history of digital computing has seen many pertinent developments. The 10 year history of computer driven plotting equipment and software is of direct interest in teaching the interrelationship of the several attributes of heart function. The initial application of computer driven plotters was directed towards static images, whether cathode ray tube or electromechanical plotter. Later, the effectiveness of programs and plotting equipment improved and production of multiple frames became economically feasible. Initial dynamic presentation applications were directed almost entirely to the hard sciences. Lately, a small percentage of applications have been developed for the soft and social sciences.

The very recent history of programmed or computer aided instruction (CAI) also lends some insight as to how one can assist the teaching cardiologist. Current CAI practice labels a computer as anything from a simple sequencer to a large scale time shared system. Most visual teaching material has been prepared for conventional teaching methods. However, when based on properly organized material, learning results have consistently validated the concept of programmed instruction based on visual input by qualified teachers (teaching cardiophysiologists).

The initiation of cooperative heart study activities between The Aerospace Corporation, and the Cardiovascular Research Laboratory, Loma Linda University School of Medicine, in the Spring of 1967 placed these disciplines in direct view of each other, and under circumstances which provided a continuing exposure to the latest developments in CAI. This study activity represents a cooperative effort by both organizations to determine whether the capabilities of the respective disciplines can be combined to provide both improved methods for teaching heart function, and to alleviate demands placed on the time of the over-burdened teaching cardiophysiologist.

Initial static image production

Concept

The Aerospace Corporation has supported perspective computer graphics investigations from company funds since 1964. On initiation of this cooperative heart study, it immediately became apparent that these capabilities should support some research applications in visual portrayal of cardiac phenomena. A pictorial representation of the exterior surface of the human heart and great vessels was selected, both as a first experiment, and as an application suitable for developing basic techniques and providing broad experience. Following the terminology of Fetter,[1] there was no formal aims definition other than to achieve a reasonably satisfactory static image with hidden lines removed, and there was no overall communications design.

Specimen preparation and film data recording

Successful data transcription required an innovative approach based on the unusual application of standard techniques.[2] The primary anatomic data was obtained from a post mortem heart; a series of 6 post mortem hearts was used to develop overall data transcription techniques. The particular heart serving as the final model was free from congenital or acquired heart disease.

Each blood vessel leading to or from the heart was cut perpendicular to its long axis at approximately 2 cm. from its junction with the heart chambers. The specimen was washed completely free of clots and inspected for evidence of structural abnormality. All heart chambers were filled with small pieces of plastic foam nearly transparent with respect to diagnostic x-ray wavelengths. This foam distended the chambers so that the heart resembled its relaxed (diastolic) configuration. The specimen was next positioned in a specially fabricated square plexiglass mounting container 10 cm. on a side, and fitted with two parallel 10 cm. registration pins in diagonal corners 1.0 cm. from each edge, as illustrated in Figure 1—Post Mortem Heart Specimen. The container was open on the top and front surfaces to facilitate attaching heart suspension threads to mounting holes bored in each surface of the container. Once firmly in the container, the heart was ready for x-ray photography to record detailed anatomic information.

The specimen and mounting container were placed in the beam of an x-ray polytome shown in Figure 2—X-Ray Polytome, which records x-ray images at pre-selected vertical intervals. This was achieved by synchronizing the opposite motions of the polytome x-ray tube and film holder, so that only a narrow plane perpendicular to x-ray tube centrum/x-ray film axis remained in focus. The apparatus was then adjusted to produce x-ray pictures of a series of these planes at 5.0 mm intervals from the upper to lower surfaces of the specimen and container. The equipment then produced from the heart specimen a series of x-ray pictures depicting chamber walls, the relative position

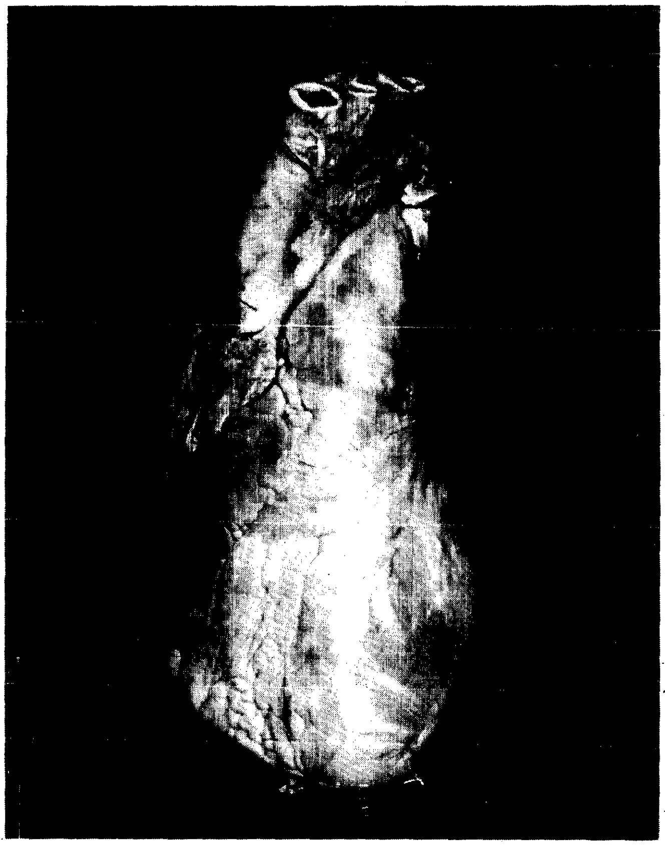


Figure 1—Post mortem heart specimen

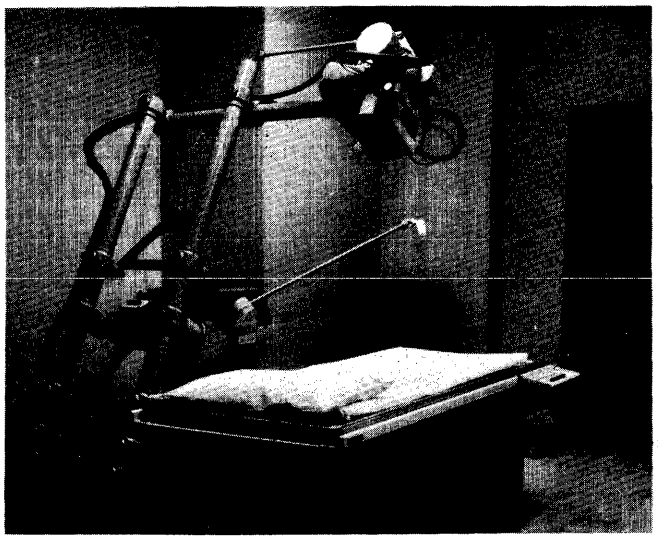


Figure 2—X-ray polytome

of one heart chamber to another, and registration pin images for picture orientation.

Data transcription

The polytome pictures or x-ray negatives are exceptionally "noisy" because of poor edge definition and emulsion noise due to x-ray scatter, which cause all but limited details to be out of focus. Thus, the next step was for the noise due to the above sources to be "filtered" by subjective exercise of the physicians' past experience during examination of each negative in the labeled sequence. This was accomplished by applying his knowledge of heart structure during the process of "edge tracing" to convert very noisy section data to "filtered" clean sections clearly depicting the outlines of heart chamber surfaces, heart valves, and chamber wall thickness.

Each individual x-ray negative was positioned on a light table, and a transparent sheet of mylar was appropriately registered. Mylar registration was of particular importance, in order that level to level relative alignment could be maintained when the outlines were transcribed to polar coordinate form. The heart chamber boundaries, external heart wall, and registration marks were traced by the physician onto the mylar overlay with an india ink pen. A resentative tracing is shown in Figure 3—Mylar Section Outline. Each mylar overlay was coded relative to its associated x-ray negative, so that the sequence was maintained in proper order. The anatomic data (surfaces of the heart) was thus transformed to outline form, and transferred to a media which would facilitate its specification in polar coordinate form during the

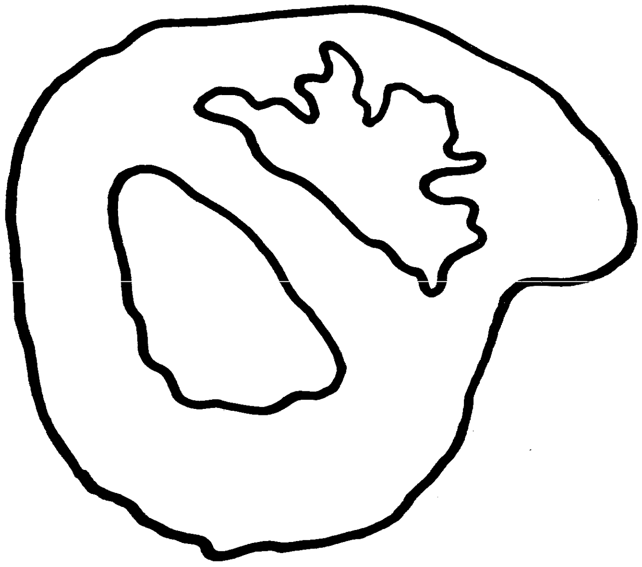


Figure 3—Mylar section outline

next step in the data transcription process. Using the registration marks for alignment, the set of mylar tracing edge outlines could be seen to provide a three-dimensional reconstruction of the heart. That is to say, they formed a Serial Heart Atlas. A reconstruction from selected Atlas sections is pictorially represented in Figure 4—Mylar Heart Surface Reconstruction.

It is of substantial importance to note that this data transcription process was achieved without damaging the original heart specimen. Medical ethics were maintained, new research data were developed, and the specimen was returned to the Pathology Labora-

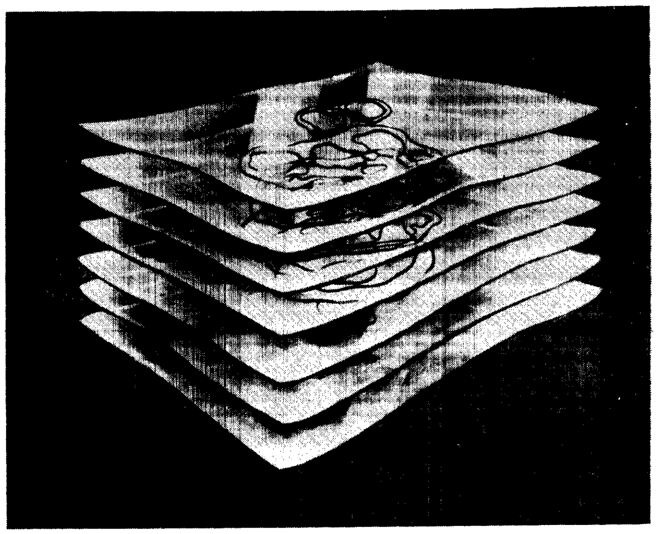


Figure 4—Mylar heart surface reconstruction

tory where the remainder of the routine autopsy was carried out. Also, the tissue specimens obtained during further medical analysis substantiated the fact that the post mortem heart utilized was free from structural disease not only on the gross, but also on the microscopic level.

The Atlas of mylar tracings was then leaved through rapidly several times to obtain a qualitative impression as to which vertical line would best serve as an axis for aligning the sequence of polar coordinate plots. A line connecting the center of the pulmonary artery and the apex or bottom tip of the heart was selected for this purpose. Each coded piece of mylar was then taken in turn from the sequence and mounted on a light table. A sheet of polar coordinate paper was then registered in an appropriate position relative to pulmonary artery/apex line and the mylar registration marks. Thus, each "non-destructive slice" from the specimen, as represented by a unique mylar edge outline sheet, was oriented in orthogonal 3-space by its z value (on the axis formed by the origin of the several polar coordinate systems) and by maintaining within each mylar plane a constant relationship between $\theta = 0°$ and the specimen box registration pins.

The first level selected was at the bottom tip or apex of the heart, and sequential levels were at 0.5 cm. increments until the top of the pulmonary artery was reached. Outlines for the main body of the heart were transferred to polar coordinate paper oriented about the pulmonary artery/apex axis. When the great vessels began to appear at the top of the heart, the multiple surface capability of the perspective plot package was utilized. The pulmonary artery edge outlines were transferred as a separate surface, but with a base level equal to the z value of the top surface of the heart and zero offset relative to the pulmonary artery/apex axis. The aorta, superior vena cava, and pulmonary veins were each treated in a similar fashion, except that appropriate x, y offset coordinates were specified relative to the pulmonary artery. At this point, data transcription was essentially complete. It remained to arrange the data in that format most convenient for processing by the perspective package driver.

Perspective plot program

The operating program used for initial efforts implemented the algorithms described in "The Perspective Representation of Functions of Two Variables" by Kubert, Szabo, and Guiliari.[3] Basically, the function to be plotted, (f) (x, y), is given in terms of a set of rectangular coordinates (x, y, z) where z = f (x, y). As in the present case, non-rectangular inputs may be utilized; they are processed internally into rectangular values. Visibility of a rectangular surface formed by evaluating the function of x_i, y_i, is developed by forming a grid connecting all equal values of x and all equal values of y. For non-rectangular point arrays, a grid is formed, and points connected for all equal values of z and all equal values of θ. Standard trigonometric methods are followed for selecting a viewing point and projecting the grid structure thus formed to a desired projection plane. The initial version of the program applied visibility test criteria between each point and any other points in the array near the line of sight. If more than one array (surface) is present, the tests were applied along the line of sight in each additional array. After completing visibility testing, the grid structure identified earlier was formed by connecting all visible points. This entire process is as illustrated in Figure 5—Heart Grid Projection, which depicts the specimen, grid overlay viewing point, and project on plane and image.

Limited core availability of the IBM 7094-II constrained array size. Thus, "square" arrays of function evaluation points were utilized wherever possible. These factors were taken into account when selecting the location line for parting radii for a numerical dissection by animation experiment, and also when selecting the number and orientation of radii to be used in specifying each divided specimen thus obtained. Thus, each polar coordinate level was represented by a z value, θ_{origin}, $\Delta\theta$, number of radii, and ρ value list for the first section, and a similar set of parameters

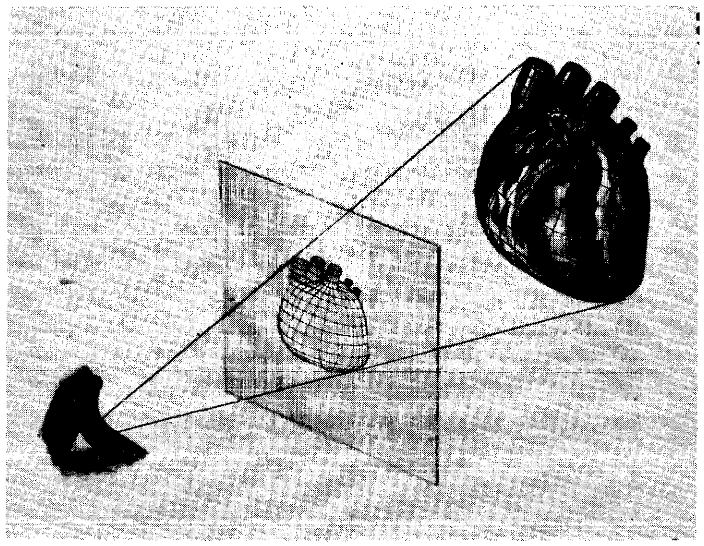


Figure 5—Heart grid projection

for the second section. The external surfaces of the heart, and the external surfaces of each of the great vessels were then each divided into that number of subsections which would best satisfy the square array condition. These measures reduced the exterior surfaces of the heart and great vessels to a collection of ordered polar coordinate points (ρ, θ, z) which were listed in an appropriate format and converted to a data deck. This method of representation resulted in 14 subsurfaces and 780 mesh points.

Initial results

The first heart images were obtained during July-August '67 and required 45 minutes of IBM 7094-II time, later reduced to 20 minutes. Examples of these experiments were plotted on an IBM 1627 (CalComp) and are illustrated in Figure 6—Closed Heart, and Figure 7—Numerically Dissected Heart. In general, the plot routine performed as expected, and provided useful imagery even though minor anomalies resulted from the visibility algorithm and the data transcription process. At times, a normally invisible line would be seen at the tip and would result in spurious lines being drawn until it crossed the next boundary. The data transcription process did not provide complete closure between the bases of the several great vessels at the top of the heart, and the resulting gaps allowed portions of the inside of the heart to become visible. However, the results were sufficiently representative of heart structure to establish the future usefulness of computer animation, provided some improvements could be made in overall perspective plot program computing time. A continuing liaison effort was initiated between the Cooperative Heart Study and the Aerospace computer graphics research activities to both observe current progress and be well aware of any impending developments which might result in more efficient image computation. Also, additional activity was initiated to eliminate data transcription noise and obtain a numerical description of a smooth surface heart. Finally, planning was undertaken for production plotting on the Waveform Display/Analyzer, a high speed, interactive, precision film scanner/recorder having a pin-registered 35 mm film transport.

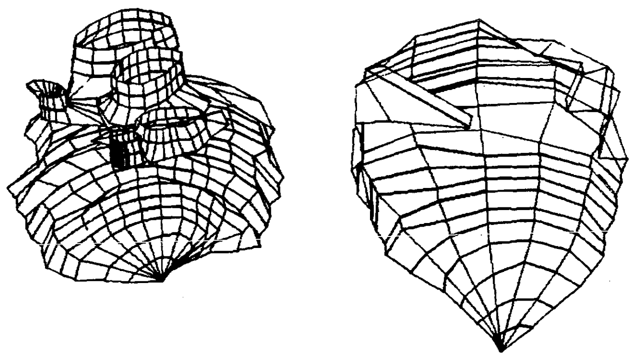


Figure 7—Numerically dissected heart

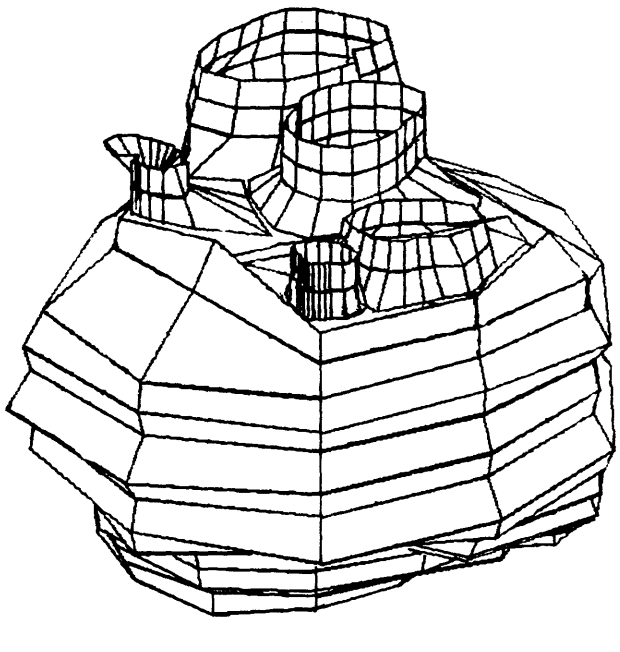


Figure 6—Closed heart

Improved image computation time

During the latter part of 1967, the visibility test algorithm was substantially revised[4] in that the projection plane was divided into a rectangular grid, and object mesh points were projected into these projection plane sub-elements. This enabled visibility testing to be limited only to the immediate area of the projection plane sub-element, i.e., was the projection plane sub-element occupied by another object mesh point and if so, which of the two competing mesh points was closest to the observer? This new algorithm resulted in roughly a geometric decrease in perspective program running time, since it was no longer necessary to compare each object mesh point to all points where the line of sight crosses the grid in the object or objects under consideration. It yielded IBM 360/65 frame times of approximately 20 sec.

As before, occasional image anomalies became apparent as illustrated in Figure 8—Closed Heart—

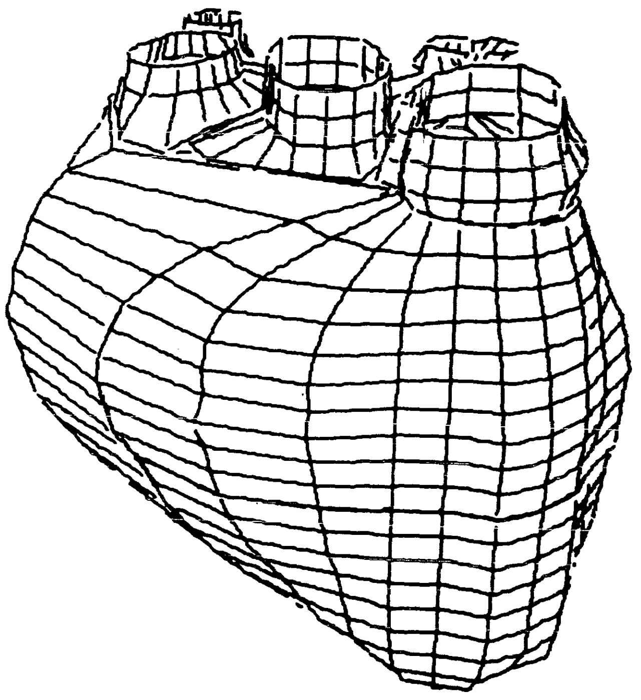


Figure 8—Closed heart—new method

New Method. Base gaps at the top of the heart still allow projection of small portions of the interior. However, line drawing anomalies now resulted primarily from line segment closure errors between adjacent projection plane sub-elements. Limitations on the number of line segments allowed to occupy a projection plane sub-element, and on the amount of closure testing, would occasionally result in short erroneous connections or "hanging stubs." Certain algorithm improvements were able to decrease this type of visual noise. At this time, it was postulated (and later substantiated) that dynamic motion would tend to effectively mask all but a small number of such anomalies, and that those remaining would be sufficiently obscured to not visibly detract from the overall image quality.

Heart beat algorithm

Basic movement data

Perspective plot package probable compute times and plot quality were now such as to justify development of a heart beat algorithm. Perturbation of the static heart surface image so that a dynamic (cine) image sequence would realistically depict the beating human heart from any viewing angle represented a significant challenge. A solution to this problem was achieved by a combination of data transcription techniques, computer graphics, innovative utilization of cardiac research instrumentation, and most importantly, something that can be described no more explicitly than the creative interplay between experience in cardiology and experience in data reduction.

The starting point was the selection of an electrocardiogram (EKG) from a healthy heart beating at a rate of 60 beats per minute. A bar plot depicting the cycle times of the right and left atria and right and left ventricles in the overall cardiac sequence was then constructed to the same time scale as the EKG. Chamber movement within these cycle times was derived from two primary sources: cineangiography and ultrasound. Cineangiography data is provided by high speed motion pictures obtained from x-ray/image amplifier pictures of the heart pumping blood containing blood-soluble radio-opaque contrast media. Such high speed x-ray motion pictures are then analyzed by a cardiologist for qualitative aspects and, by measuring distances on projected images for qualitative information. The measured dimensional changes are plotted against time. Ultrasonic techniques provide independent determination of the same spatial data, i.e., variation of chamber dimensions with time. Sound in the 2.5 mh. range is generated at a prf of approximately 1 kh. The transducer is placed on the chest over the heart, and the returns are displayed in a spatially calibrated A-scope format, which allows direct observation and determination of dimensional changes. Spatial displacement determined according to these methods was then folded with the chamber cycle times in the overall cardiac sequence to produce the plot of volume vs. time for atria and ventricles shown in Figure 9—Heart Chamber Volume Sequences.

Chamber static and dynamic mapping

Conversion of this two-dimensional summary expression to a perspective portrayal of three-dimensional distributed movement required mapping of the atrial and ventricular areas into the grid representing the external surface of the heart, and establishing appropriate vectors for each point so mapped. Chamber area mapping was accomplished by hand sketching outlines on two static perspective plots. After validation, the data were then mapped onto 80 column free form coding sheets. Map resolution was increased by assuming a variation in acceleration and magnitude of distance moved between the center and edge of a chamber area, with the central region having the highest acceleration and the edge the least. Thus, each external heart surface grid point was assigned a chamber and type designation, and a multiplier was assigned to each type and applied as appropriate. Specification of direction, acceleration, and magnitude for grid

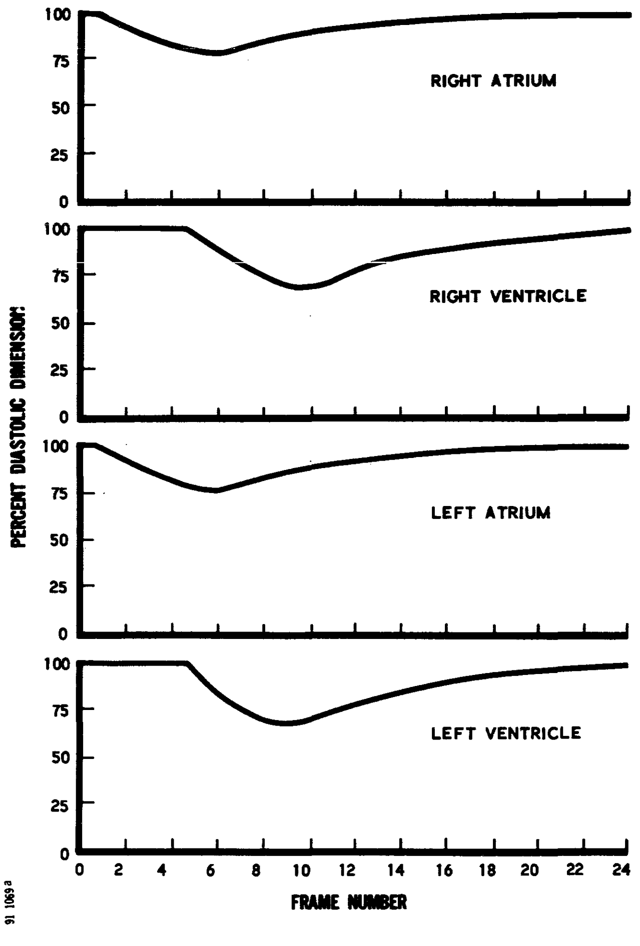


Figure 9—Heart chamber volume sequences

point movement was simplified by allowing only two-dimensional variation. In particular, this variation was between the grid point and the polar coordinate origin, or, along ρ. Thus, for any particular time in the overall cardiac cycle, the appropriate displacement values for the four chambers were picked, and then folded with the appropriate type multiplier, depending on whether the grid point under consideration was in the center, mid region, or edge, of the chamber under consideration.

Generalized motion algorithm

Basic surface description method

Although not immediately obvious, the methodology used in describing the heart surface and heart motion is quite representative of a more generalized capability. A variety of data transcription techniques may be used, as long as they result in a systematic description of the surface in orthogonal 3-space. The present heart technique of displaced polar coordinate sections provides satisfactory results. Function evaluation at specified intervals also provides usable data. Isometric drawing data can be transcribed so as to provide the basis for logically correct object surface descriptions. In each of these representative data transcription methods, as well as other similar ones, almost any data transcription or data generation method will work, as long as it may be represented by some valid algorithm.

The input data are then processed by a driver to create a properly organized grid structure ($P_{x,\ y,\ z}$) data set.

Basic motion description method

Computer animation depends upon systematic perturbation of a basic logical entity resulting from data transcription/driver output. The systematic perturbation may result from either analytical function evaluation or from empirically derived data. Heart motion was empirically determined by measuring spatial displacement through ultrasonic pulse displacement with time, and direct measurement from cineangiography. Another logically similar method originated by Fetter is based on transcription of landmark (e.g., shoulder, elbow, wrist) movement from the photographs of Muybridge.[5] Identical methodology may be applied to other dynamic inanimate phenomena not conveniently expressed by analytical notation. In each of these cases, similarity to surface description methods exist, in that a driver is used to assemble the basic landmark movement in orthogonal 3-space, and also interpolate as required for mapping into the object surface description and for perturbation at the selected framing rate.

Motion animation

Availability of an object surface description in the form ($P_{x,\ y,\ z}$), and motion specification in the form of properly interpolated empirical description of landmark position variation with time, allows computer animation to proceed. If the object surface description and landmark motion drivers are written in generalized form, they may be coupled to the animation driver, and animation may be accomplished at any desired rate. If not, all motion perturbation must be related to silent or sound projection rates on an a priori basis.

It should be noted that all of the preceding GENERALIZED MOTION ALGORITHM discussion is concerned solely with perturbation aspects of computer animation. Perspective, hidden line removal, list processing organization for "save" procedures, etc., are separate subjects.

Feasibility test film

Validation of the algorithm was undertaken by selecting a non-rotating, beating view at a heart rate of 60 beats per minute. This parametric combination allowed a full heart beat cycle to be completed in one second. Thus, the number of plots was determined by the standard sound projection rate of 24 frames per second. The perspective plot package drivers, which now consisted of the heart surface formatter and perturbation algorithm, were set up for 24 plots. After computation, the plots were produced by an IBM 1627 (CalComp) using india ink on white paper. The plotting time of several minutes per image (frame) was acceptable for only the shortest animation sequences. For validation purposes, however, the availability of a large, high contrast, hard copy image supported easy visual error checking. The cyclical characteristic movement of a beating heart allowed efficient use of a single sequence, in that it minimized the amount of raw plotting required. Repeated (film) printing provided unlimited viewing time to check all aspects of perturbation algorithm fidelity. The automatic scaling and centering algorithms within the perspective plot package were also evaluated for contributions to overall image quality.

Inspection of the actual plots which were to serve as the basic animation cells immediately disclosed excessive image centering movement. This required alteration of the centering algorithm for subsequent animation efforts and, in the validation case, frame-to-frame registration on the two vessels having maximum displacement between each other, i.e., the pulmonary artery and superior vena cava, as the individual plots were cut and punched to standard animation cell format. Static and "flip" evaluation of the cell booklet indicated that an approximately correct portrayal of a heart beat had been achieved. Finally, labeled overlays and titling were prepared on clear acetate.

Cine footage was obtained by standard animation photography procedures. Double printing (2 sequential exposures) of each frame supplied half-rate motion to allow longer times for dynamic error check observation. Multiple sequences of standard single printing supplied footage for standard motion validation.

Overall evaluation of the feasibility test film disclosed that the basic perturbation algorithm was logically sound.[6] The only motion anomalies resulted from two individual grid point mapping errors (out of 780), and a data transcription error in specifying right atrial movement. As expected, minor grid line direction plotting errors produced by the new visibility check algorithms were largely masked by the dynamic movement. The simple 3 minute film was a happily unexpected success of major significance. An algorithm had been developed for accurately expressing natural physiological motion in numerical form, thus opening the door to high speed computing synthesis of natural motion by computer animation.

Experimental teaching film

Communication design

During the 6 to 8 weeks that were required to develop and validate the perturbation algorithm, communication design had been occurring. A script outline had been developed for a film to disclose technique capabilities and education possibilities to cardiologists and physicians. Animation requirements involved illustrating dynamic portrayal of heart surface smoothing, 360° rotation, tipping about the vertical axis at classical viewing positions, and computer synthesized dissection to allow viewing of interior heart structure. As in all manual or computer animation activities, each scene outline was reviewed to reduce to a minimum the number of raw stock animation frames. At this point, a highly detailed animation script was prepared, which completely specified all data necessary to compute each individual frame of raw stock animation. This information was also required to enable correct sequencing of raw stock animation in assembling the complete scenes.

Production computer animation

Computer run output was produced on tape suitable for driving an IBM 1627 (CalComp) incremental plotter. This resulted from the fact that the Aerospace (San Bernardino Operations) Mathematics and Computation Center has an extensive investment in plotting systems and applications programs, and production experience historically traceable to the incremental plotting systems programs developed at STL by K. G. Tomikawa and J. R. Blackmer. This type of output, while not of optimum efficiency, was indeed satisfactory, and had the substantial advantage of providing a useful run validation mechanism through selected hard copy plots.

The script outline required several thousand frames of computer animation. The IBM 1627 (CalComp) incremental plotter tapes were converted to a format suitable for driving the Waveform Display/Analyzer (WD/A), a high speed, computer driven (IBM 1800), interactive, precision, film scanner/recorder.[7,8] Of particular importance, this system possessed both a highly accurate pin-registered 35mm camera, and a real-time interactive console for monitoring plot production. The translation program provides: (a) simultaneous film recording, and a tape of WD/A instructions, or (b) a tape of WD/A instructions only. The first capability is normally used for film image validation of raw animation stock frames. The second is used for high speed plotting at essentially tape read speed, when raw stock frames are assembled into scene sequences.

It is interesting to note that the second capability was immediately developed after the first raw stock frames were produced. Selected frames had been validated on the IBM 1627 (CalComp) plotter, the tapes translated and a 35mm film master exposed on the WD/A. Two days of desk effort plus two days of printer time were required to produce a script usable by an optical printer operator and for production of 16mm master footage. This immediately indicated the preferability of using standard tape-to-tape copy, i.e., *tape to film copy* methods.

Approximately 2 weeks of elapsed junior programmer time developed an IBM 1816 (typewriter) conversational tape-to-film animation sequence assembler. A simple tape search program and capabilities for an elemental dialogue concerning titling and plot frame identity allowed the two days for script production and two days of effort for producing a 16mm master animation scene sequence to be reduced to approximately 2¾ hours WD/A plot time plus a straight 35mm to 16mm reduction run. Thus, it was now possible to eliminate optical print script preparation time, and to provide the optical printing operator with a reduce and copy job, eliminating time consuming optical printer sequencing.

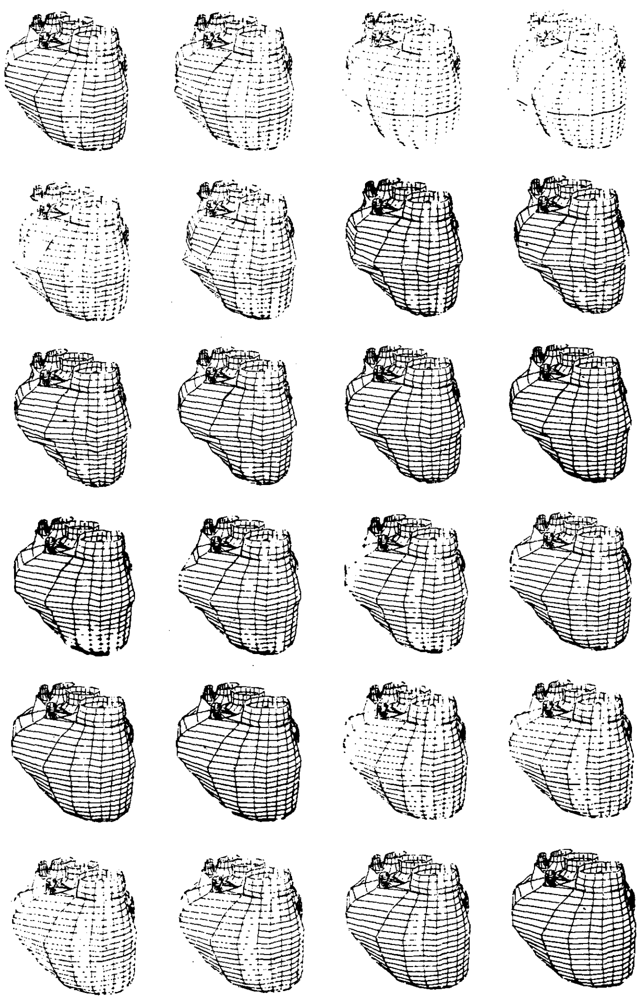


Figure 10—Complete heart cycle

Film tutorial

During production of the animation footage, live action footage had been shot for the remainder of the script. Communication design had been based on an aims definition to provide an experimental film outlining teaching capabilities which could be implemented with this new technique. The first version of the film was entitled "Heart Motion by Computer Graphics".[9] It was a medically oriented tutorial which outlined the capabilities of digital computers, incremental and CRT plotters, computer graphics perspective plot programs, anatomic data acquisition techniques, basic animation, and examples of computer animation and teaching situations. A full 24 frame sequence depicting one complete beat is illustrated in Figure 10—Complete Heart Cycle. The film was directed toward medical students, post-graduate physicians and patients.

Initial teaching experience was obtained in a variety of situations: (a) laymen were shown the short feasibility film at an Aerospace open house, (b) the physician co-author used both the short feasibility film and the "Heart Motion by Computer Graphics" tutorial during his teaching activities at the Loma Linda University School of Medicine, and (c) both senior authors have made a variety of presentations to selected physician and engineering groups.

Two significant reactions were obtained from this early teaching experience: (a) medical exposure established teaching potential, particularly when combined with simultaneous presentation of associated cardiac parameters, and (b) the medically oriented scenes and narration were not suitable for engineering or programming evaluation. Therefore, additional live action scenes were developed to emphasize data conversion and computer programming aspects of the problem.

A second, engineering oriented film, "Teaching Heart Function by Computer Animation" was prepared.[10] These two films have served to adequately present salient technical and teaching factors to both types of audiences.

Advantages and disadvantages

The unique nature of this type of material has stimulated forward looking teaching physicians. Particularly, those with sufficient exposure to advanced instrumentation and computational methods to produce an appreciation of the substantial teaching advantages which could be provided by audio visual material based on computer animation technology. Such technology enables the cardiac educator to readily develop any desired static or dynamic normal or diseased heart condition. A portrayal may be developed without waiting for the right type of cardiac condition to become available for photography through open heart surgery. Similarly, there is no need to place impossibly large animation requirements upon the already overloaded medical illustrator who normally is much more of an anatomically trained and highly skilled detail technician, rather than a proficient animator. High speed computation allows the production of animation from any desired viewing point with equal ease of illustration, once the basic data transcription has been accomplished. Of particular interest, is the fact that "numerical dissection" techniques allow internal dynamic viewing of any "heart preparation" desired; this type of presentation cannot be accomplished by any means other than manual or computer animation. Computer animation techniques also greatly facilitate the presentation of any desired combination of cardiac structure and associated functional attributes. Thus, the teaching physician can, with a minimum of effort, direct the production of an incremental visual presentation of increasing complexity, and where required, increasing functional rate, all with appropriate audio on the sound track.

Audio visual material does not have the personal contact of the live classroom and does not allow for the comprehensive stimulation of fast moving question-and-answer dialogue. However, well developed instructional material of the type outlined in this paper can summarize and greatly condense the presentation of significant material from many hours of classroom and laboratory work into two half-hour films or tapes. It should be strongly emphasized that the techniques of computer animation are particularly adaptable to utilizing experience gained over the last few years from programmed and computer aided instruction. And, that such films could be used for audiences ranging in size from a single student or physician to a hundred or more at any time during the day or night.

Current activity

Current activity of this aspect of the Cooperative Heart Study is devoted to planning teaching system development requirements. Immediate effort concerns accomplishing the additional data transcription necessary to specify the atria, ventricles, and valves. Subsequent data transcription will provide that data in numerical form necessary for introducing electrocardiograph plots, ultrasonic cardiograph plots, pressure, and flow. The last data transcription activities will be directed toward introducing visual presentation of heart sound data.

Parallel activities concern additional computer graphics research for achieving perspective presentation without the grid structure, but retaining sufficient surface and edge detail to provide adequate visual cues. Similarly, the present perspective plot package driver is being evaluated to determine what changes would make it more usable for high volume production of animation. That is to say, what data and control information formats will allow minimum time requirements for input deck preparation.

Other medical investigations are being devoted to outlining a catalog of required normal and diseased presentation and, in particular, the explicit content and sequence of these presentations. These medical teaching studies are also considering whether or not computer animation can best be applied to teaching heart function through group orientation or by individual consoles. Finally, if individual consoles are to be used, should they be film or video tape oriented and, should they provide only an ordered presentation, or should they allow random sequences based on student response?

Other possible applications

Finally, some limited consideration is being given to the identification of needs, and the analysis of the probability of success for extending computer animation techniques to other medical fields such as pulmonary studies or teaching obstetrics. The generality of the algorithm for accurately expressing natural physiological motion in numerical form (as earlier independently conceived by Fetter and as outlined in this paper) provides for high speed computing synthesis of natural motion by computer animation, and thus allows realistic identification of solutions to such teaching problems.

SUMMARY AND CONCLUSIONS

The historical development of a computer animation technique for teaching heart function has been reviewed. Of particular importance, early planning activities were organized to be ready to take advantage of future solutions to clearly identified problems.

The initial feasibility test film validated the perturbation (heart beat) algorithm. Most importantly, it established a basic algorithm for expressing natural physiological motion, thus allowing high speed computing synthesis of unlimited amounts of additional natural motion by computer animation. At this point, the aims definition, communication design, and additional data transcription and computer programming were undertaken for production volume automatic drawing (computer film recording) of heart animation. Live action footage was photographed to round out communication design requirements. The animation and live action footage were combined to produce an experimental heart function teaching film tutorial and an engineering oriented film explaining programming details. Medical and engineering acceptance accorded these films well appears to justify additional development.

The generality of basic data transcription and animation techniques allows direct application to other medical teaching activities outside the area of cardiology such as pulmonary studies, obstetrics, and other similar problems.

ACKNOWLEDGMENT

This study was supported by Aerospace (San Bernardino Operations) company funded research and NIH General Research Support Grants 05-94 and 01-08 through Loma Linda University School of Medicine.

REFERENCES

1 W A FETTER
Computer graphics in communication
McGraw Hill Inc New York 1965

2 A F BOWYER
Computer graphics simulation oj the human heart
Proc Society of Photo-optical Instrumentation Engineers (SPIE) 13th annual Technical Symposium August 19-20 1968

3 B R KUBERT J SZABO S GIULIERI
The perspective representation of functions of two variables
ATR-66(S9319)-1 Aerospace Corporation June 1966

4 B R KUBERT
A computer method for perspective representation of curves and surfaces
Aerospace Corporation 1968

5 W A FETTER
The second man
16mm sound film (1968—8 minute running time)
Available from The Boeing Company Commercial Airplane Division Renton Washington

6 A F BOWYER A H GOTT
Heart motion by computer graphics
16 mm silent motion picture film (1968—3 minute running time)
Available from the Cardiovascular Research Laboratory Loma Linda School of Medicine or Data Reduction Department Mathematics and Computation Center Aerospace Corporation (San Bernardino Operations)

7 A H GOTT
A data communication subsystem
Proc Third National Symposium Society for Information Display (SID) February 1964

8 A H GOTT
An integrated film reading and display system
Proc Seminar-in-depth Filmed Data and Computers Society of Photo-optical Instrumentation Engineers (SPIE) June 13-14 1966

9 A F BOWYER A H GOTT
Heart motion by computer graphics
16 mm sound motion picture film (1968—12 minute running time)
Available from the Cardiovascular Research Laboratory Loma Linda School of Medicine or Data Reduction Department Mathematics and Computation Center Aerospace Corporation (San Bernardino Operations)

10 A H GOTT A F BOWYER
Teaching heart function by computer animation
16 mm sound motion picture film (1968—15 minute running time)
Available from the Data Reduction Department Mathematics and Computation Center Aerospace Corporation (San Bernardino Operations) or toe Cardirvascular Research Labortory Loma Linda School of Medicine

The many roles of computing on the campus

by THOMAS E. KURTZ

Dartmouth College
Hanover, New Hampshire

INTRODUCTION

At Dartmouth College the "computer" has become an essential component of the university community comparable to the library in importance, size, and diversity of application. While such a statement no longer raises eyebrows, in few other cases has the extent of the infusion of computing into the life of the university community been as great, or as painless, as at Dartmouth. The purpose of this paper is to report some of the many ways in which computing has been found useful in teaching and research at Dartmouth, and to claim that these almost revolutionary (though almost painless) developments were generated primarily by having freely available to all students and faculty a general purpose time-sharing system equipped with a simple and easy to learn language (BASIC), and a simple and friendly user interface.[1]

A brief statement about Dartmouth College will establish the framework of these discussions. The College has an undergraduate student body numbering about 3,000 in residence. Of these, about one-fourth eventually major in one of the sciences. The other three-fourths elect a major subject from the humanities or social sciences. There are about 150 graduate students in Ph.D programs, mostly in the sciences. In addition, the three associated schools of medicine, business, and engineering claim about 400 students of which more than half are MBA candidates in the business school. The combined faculty and administrative staff of all the components of the university number about 600, of which about 250 are teaching faculty of the undergraduate college. In terms of computer usage during the school year 1967-1968, about one-half of the faculty members found it convenient or necessary to use computing in some form in their teaching or research. About two-thirds of the undergraduate students and three-fourths of the graduate students also found it necessary or desirable to employ computing in their studies.

A historical note: The birth of the first Dartmouth Time-sharing System in the spring of 1964 and its subsequent influence on the campus and in the computer world at large is fairly well-known. Not too well known is the fact that the first publicly demonstrated remote use of a computer took place also at Dartmouth College. The time was September of 1940, the occasion was a fall meeting of the American Mathematical Society, and the computer was a relay calculator designed and constructed by Dr. George Stibitz of the Bell Telephone Laboratories. Located in New York City, it could perform operations on complex numbers. A teletype machine located on the Dartmouth campus was connected to the calculator via long-distance telephone lines. The calculations took up to 20 seconds to carry out, but one reporter was moved to refer to the "lightning-like speed" of the machine. Another reporter wondered whether this machine would have any important use. While this historical event bears no direct relationship to the much later development of the Dartmouth Time-Sharing System, or to any other time-sharing system for that matter, it does point out that remote usage of a computer is not a new idea.

The many forms of computing

Instead of talking about *the* role of *the* computer in college education, we should be considering the multiplicity of roles of the many types of *computing services* in that environment. For instance, Dr. Stibitz' machine could perform arithmetic operations on complex numbers and were highly useful to the Bell Laboratories engineers who needed such calculations. But the device had no use outside this narrow area. In contrast, the modern digital computer can at one moment be a number-cruncher grinding out millions of numerical calculations. At another time it may be symbol manipulator requiring practically no arithmetic calculations. At still other times it may serve as a vehicle for managing a programmed instructional sequence. Because the word "computing" does mean different things at different times and in different contexts, it may be useful to formally recognize several fundamentally different functional types of computing. Of special pertinence to the university community are the "computer services" named and briefly identified below:

A-computing; the computing characterized typically by administrative data processing where large amounts of input data undergo relatively simple transformations to reappear as large amounts of printed output.

C-computing; calculational computing where the main purpose is to provide numerical calculations. We may wish to further distinguish between small, medium, and large calculational requirements. The amounts of input and output are usually small.

V-computing; the use of the computer as a vehicle to support CAI and similar activities. Here, interaction with the user is crucial while the input, calculational, and output requirements may all be small.

I-computing; the types of service needed to support information systems such as are needed for management decision applications or library automation. The prime requirement here is the maintenance within the computer system of large files.

T-computing; in an educational environment, one of the most important forms of computing is a "being taught" machine. The student will learn about a process by representing it as a program that works; that is, by "teaching" the computer how to carry out the process. The need here is for quick and easy interaction with the user, which implies simple command and programming languages.

A general purpose service

We could choose to define the term "general purpose computer" as one which can handle all of these different types of computing functions. In theory, specialized computing systems should be able to provide a single type of service more economically than can a general purpose system. It is a remarkable fact, however, that the Dartmouth Time-Sharing Service (DTSS) has been able to provide exceptional service to all the above types of computing needs except for conventional administrative data processing and for the very large number-crunching problems. Conventional administrative data processing requires efficient unit-record input and output, and almost none of the capabilities of the general purpose time-sharing system is needed or even useful to carry out this task. With number-crunching, any computer that devotes a significant portion of its time to this task must be considered in part a dedicated device, and perhaps even an integral part of the laboratory apparatus it serves.

For all other purposes listed, the general purpose time-sharing service as has been developed at Dartmouth College is extremely effective. It provides a superb service (i.e., response within 5 or 10 seconds) for all small computational jobs requiring, say, less than a second of computing time. (In fact, many student jobs require as little as 20 to 30 milliseconds of chargeable computer time.) If a new program is being created, there is no question that a general purpose time-sharing system greatly simplifies the debugging process.

The Dartmouth System is proving extremely useful in supporting large data bases of several types. The programs for maintaining the files in the library automation project become almost trivial in view of the general purpose services offered by DTSS. Very little additional programming effort has been needed to develop file-maintenance and user-interface routines.

While it would seem that special purpose computers are needed to support CAI, at the present time a flexible general purpose system (such as DTSS) can be an extremely useful and economical vehicle for this work. Since the use of the system is so easy, the special systems programming involved is greatly simplified. And because the costs are spread over such a large base of users, the costs of the CAI usage and program development are very small. The Dartmouth System provides general file-handling capabilities and all of the other services needed to carry out this work, except possibly instantaneous or synchronous response, since the system as a whole is asynchronous. Thus, a significant amount of experimental work in CAI is being carried out without investing in a specialized CAI rig; this factor will remain important until much more research and experimentation suggest the most economical form for CAI vehicles.

Finally, as a medium in which a student can express algorithms, the general purpose time-sharing system is unmatched. The quick response and the ultra easy modification of files at a relatively low cost ($3 to $4 per terminal hour) make possible the use of DTSS as a "being taught" machine on a large scale. It is this capability that is now being accepted as perhaps the most significant role of "computing" in education. The "computer" does not become impatient with slow students, nor does it discriminate between students; all students are treated alike, and the ultimate test is "getting the right answer."

One important category of computer services are those required for the construction, debugging, and maintenance of the system itself. DTSS provides file creation and editing services, on-line assembly service, and a sophisticated DDT (direct debugging technique) that permit systems programmers to work concurrently with the ordinary users. DTSS allows the changing of official systems (compilers, editors, etc.) "on the fly" from one of two very privileged control consoles. Since the systems maintenance is done during time-sharing itself, the system can be scheduled for longer periods of time; it doesn't have to be "taken down" more than once a week for software and file system maintenance.[2]

The Dartmouth system is the victim of the myth that it is devoted to a single language (BASIC). Actually, it has never been true that BASIC is the only language available, even with the original system developed in 1964. By the fall of that year users had a choice between BASIC and ALGØL. Later, FORTRAN and LISP were added. The present DTSS offers on-line service in BASIC, FØRTRAN IV, and ALGØL 60, and has a number of simulators for other computers such as the PDP-9. Additional services are available through foreground initiated background. The file system is random access and content-independent. A system of access controls permits multiple simultaneous use of a single file. There is a full range of editing services for both line-numbered and non-line-numbered files. String processing can be done using either the on-line string editor or a system patterned on TRAC.

The BASIC language provides for random access string and numerical files as well as console-compatible "teletype" files. While no string operators as such are included, any string can be mapped into a vector of numerical character values for arbitrary manipulation (the mapping can go in either direction.) BASIC is designed to provide the services it offers in as efficient a way as possible, so that the same offerings in the context of another language might be more costly.

One consequence of the development of DTSS has been the discovery that almost all computing jobs are small. Jobs which in other environments require days or weeks of programming time and dollars worth of computing can be done on the Dartmouth System for minutes or hours of programming time and a few cents worth of computing. For instance, statistical programs can be very simple; they do not have to provide a many-paged comprehensive report because it is so easy to modify the output on the spot to suit the individual purpose. The public library in the Dartmouth computer contains short and simple programs for dealing with multiple linear regression, analysis of variance, and other statistical applications. Another example: about ten years ago a very extensive Markov chain analysis program was prepared in FØRTRAN by a colleague. The same program written in BASIC is about 50 lines, or about one-fourth as long, partly because BASIC contains more sophisticated statements but mainly because elaborate output is simply not needed.

Another important side effect: DTSS provides a means of greatly hastening the intellectual development of both faculty members and students. Because the computer is so easy to use, practically no one avoids it. Thus, we find many students teaching themselves about programming without the benefit of even a single formal lecture, and faculty members learning about statistics and other quantitative methods, again without a noticeable investment of their time.

The geographical extent of DTSS

While statistical data do not tell the whole story, some information of this type can be useful in establishing the magnitude of the infusion of computing into the life of the College. DTSS offers terminal service primarily to teletypes, including the standard Model 33 and 35 as well as the slightly faster Model 37. Any other terminal device that uses the ASCII code and standard teletype data rates can also be used. Plans call for an implementation of the IBM 2741 terminal, which can be done very easily. Several small plotting devices are used on the Dartmouth computer through telephone lines operating at slow data rates (110 BAUD), and software for two of these have been prepared in BASIC. The system also operates several character display units requiring a standard high-speed telephone line connection. The system will also be connected to a satellite computer (PDP-9) which operates a light pen oscilloscope display unit. Since all of these devices operate over telephone lines, they could in theory be located anywhere on the telephone system and still be controlled by DTSS.

There are about 88 teletype terminals on the Dartmouth campus, distributed as shown in Table I.

In addition to the on-campus teletypes, there are approximately 52 teletypes in schools and colleges in the New England area. Most of these teletype machines are operated full-time during the day and into the night except for the public secondary schools. Table II shows the distribution of these terminals.

Table I—Teletype locations at Dartmouth
(February 1969)

Location	Number of Terminals
Kiewit Center Public TTY Room (Students)	16
Tuck School of Business	11
Thayer School of Engineering	8
Medical School	4
Departments of Arts and Sciences	20
Computation Center "in-house"	19
Miscellaneous (In research projects, homes, etc.)	10

Table II—Location of full-time teletypes in outside educational institutions

(February 1969)

Colleges	Number of Terminals
Bates	2
Bennington	1
Berkshire Community	2
Bowdoin	1
Colby Junior	1
Middlebury	2
Mount Holyoke	5
New England	2
Norwich University	1
Vermont Technical Institute	2
University of Vermont	1
Windham	1
High schools	
Private (12 schools)	16
Public (10 schools)	15

At the present time, most of the teletypes operate through the regular Centrex switching network using 103-type datasets. Some of the teletypes in the public teletype room are non-switched and use the much cheaper 109-type dataset. Most of the teletypes on the switching network can access the computer through single-digit dialing. Although most of the teletypes are switched, it is common for a department or an outside secondary school or college to dial in at 8 A.M. and remain on throughout the day.[2]

The Dartmouth System is designed to handle between 120 and 150 users simultaneously. The largest number of simultaneous users actually achieved under normal conditions was 113. (This occurred in January of 1968 using the earlier Phase I system, and included a large number of commercial users under the joint-use arrangement with General Electric in effect at that time.)

Although anything approaching an exact count of the number of users of the service is out of the question, an analysis of valid user numbers against which charges were made shows that over 8,000 "persons" used the Dartmouth computer during the period 1 October 1967 to 30 June 1968. During one month alone (May 1968) over 4,000 "persons" used the system. Table III[1] shows a breakdown of these figures. There is a strong reason to believe that the number of different users is somewhat higher than 8,000 since it is common practice in some schools and colleges to reassign user numbers during the year.

Table III—Size of user community

	Number of Different Users	
Group	May 1968	Year (9 months) 1967–1968
Dartmouth		
Undergraduate	1,000	2,000
Graduate	250	450
Faculty	100	150
Other colleges	550	950
Secondary schools	2,200	4,600
TOTAL	4,100	8,150

The average use level for the 140 teletypes located throughout Dartmouth and the participating schools and colleges is one-half to two-thirds.

Table IV shows more detailed usage statistics for one week during January of 1969. (The system in operation during the period shown was the earlier so-called Phase I system but with no GE commercial customers.) Slightly higher usage levels are experienced near the ends of the terms.

In order to save somewhat on long-distance line charges, one experiment using a multiplexing apparatus has been carried out and another is in progress. The first experiment with Mt. Holyoke College shows that the communications and teletype costs for five terminals at that distance can be as low as two terminals using ordinary non-multiplexed service. Table V shows the approximate cost breakdowns.

Table IV—Users logged on as a function of day and time of day. Period covered:

27 January 1969 to 1 February 1969, 8 AM to 6 PM.

Time	Mon	Tue	Wed	Thur	Fri	Sat	Average
815	35	41	14	30	40	18	30
915	54	66	49	49	56	45	53
1015	52	76	67	63	62	58	63
1115	60	75	73	71	80	66	71
1215	56	61	67	39*	67	53	57
1315	55	59	74	61	0*	74	65
1415	72	59	78	77	0*	71	71
1515	68	73	74	84	69	70	73
1615	78	74	71	71	69	70	72
1715	58	68	55	63	63	55	60
average	58	65	62	60	63	58	61
peak	78	74	78	84	80	74	

* effects of severe crashing during these periods.

Table V—Approximate cost analysis showing economies using a simple multiplexing device over a distance of 75 miles.

Standard approach	
Teletype (model 33, rented)	$45
Datasets (2 103-type)	50
Long line (75 miles at $3 per mile)	225
Cost per teletype per month	$320
With Multiplexer	
Multiplexer (purchased)	$9,000
Teletypes (5 purchased)	3,000
Cost per month (3 year life)	$ 333
Maintenance (estimated)	150
Long line	225
Datasets (2 202-type)	100
Total	$ 818
Cost per teletype per month	$ 165

The development of time-sharing has greatly accelerated the emergence of the service and utility nature of computing. A computation center is now less concerned with selling time on the computer than with providing computational services of all sorts, and its success depends on the quality of the services provided. Reliability and time availability are decisive in affecting the way the users view the quality of the service. The Dartmouth Time-Sharing Service is scheduled to operate from 8 A.M. to midnight, seven days per week. Being even two or three minutes late is viewed as a serious matter.

In this early stage of development of such a system reliability is a problem. Not only do numerous software bugs still exist in all such systems, but air conditioning and power failures are noticeably frequent. It is unfortunately still meaningful to measure the reliability of systems in terms of the crash rate. We view a crash rate of once per day as barely acceptable, and a crash rate of once a month as nearly ideal.

Impact on teaching and research

The most important result of the Dartmouth project is the use which the students and faculty have been making of it. Very early in its existence, students discovered that they could have the computer relieve the tedium of calculations required in homework exercises in many areas. Instructors then began asking students to solve a class of problems by writing a program. A large number of students have elected term projects and longer-range projects involving extensive use of the computer. Students in the laboratory sciences have discovered that the computer is extremely useful in reducing laboratory data, and in at least one case the teletype machine has been moved into the laboratory itself. On the lighter side, the students enjoy using the computer for fun things such as playing a simulated football game or demonstrating to their weekend dates their "prowess" on the computer. All these activities of course contribute to the liberal arts student's understanding of computing. The use by students is completely open. There is no attempt to control the uses they make of the computer; in fact, there is almost no way of finding out what they are doing except by walking into the teletype room and looking over their shoulders.

The student is not charged directly for his computer use; the charges for all the students are sent as a single bill to the College at the end of the year. Budgetary control is thus not exercised upon the individual student user but is exercised informally on the group as a whole through such indirect means as the control of the number of teletype machines available to the students and the level of computer service available to them without special additional privileges having been granted.

As with students, all faculty members who have a need for computing in their work have easily found their way into the system. In addition to the obvious research applications of the computer that are well known and common to all universities, additional services either provided for or made more easily obtainable in a general purpose time-sharing system are many. For instance, the instructor can prepare last minute examples for use in his lecturing. He can check conjectures concerning mathematical models or theorems. He can perform small calculations of temporary interest. Many of these examples of aid to teaching and research have been very small, and as such do not warrant special publication. But it must be very personally satisfying to spend several minutes of terminal time to save perhaps several hours of precious research time. Such benefits become highly important when multiplied by the number of faculty users.

Classical computational problems as statistical data reduction become much simpler on a general purpose time-sharing system. Unless the data themselves are extremely extensive, the ordinary statistical calculations are very short and permit the user to test, sequentially, several conjectures about his data at a single sitting.

Simulations in management gaming become routine in a general purpose time-sharing system. In one case, a marketing game was prepared several years ago for use in the School of Business Administration. (The programming was done by the professor's son, who was a high school student at the time.) Students in many courses are routinely assigned simulation problems. It is now very simple in the elementary statistics course, for instance, to have the students investigate the behavior of the t-test under various kinds of non-normality.

The availability of the Dartmouth Time-Sharing Service has spawned several local CAI projects. These have been carried out by the instructors actually teaching the courses involved. The programs were designed to be useful in the course, were prepared and written entirely by the instructor, and were tested on the students while the course was in progress. One program teaches elementary facts about climatology in the beginning geography course. The instructor claims that the use of this baby CAI program saves him two weeks of lecture time and does the job more effectively than he had previously done it. Several very extensive language-drill programs have been prepared by two different members of the Spanish Department. Again, the programming and design was done entirely by the instructors themselves. It should be emphasized that these are drill rather than teaching programs, but the cost for their creation and subsequent use by the students is extremely small. Of course, if you have a program that works for Spanish, it can be made very easily to work for German or French or any other language. These drill exercises on the computer are being expanded to include other languages. This work is being done in cooperation with instructors at Dartmouth and in the schools and colleges cooperating with us.

Besides the role of the time-sharing service in individual teaching and research, it is an ideal medium upon which to develop information retrieval systems dealing with large data bases. The IMPRESS project[4] is developing an easy-to-use interface through which social science faculty members and their students can experiment and interact with real data on a daily basis. Since the time-sharing system provides most of the capabilities needed (file storage and maintenance, and terminal service), the additional systems programming effort needed is small. More time was thus available to spend on the design of the interface and the optimal structuring of the data for student use.

Planning and program budgeting are rapidly becoming essential for institutional solvency. A program for making broad-brush budget projections based on a suitable model of the institution is almost trivial. Furthermore, it can be invoked during administrative staff meetings to quickly judge the long-range financial effect of plans under discussion. This application has been implemented in DTSS for several years. The model grows in complexity each year as the comptroller *personally* adds to his program.

Another important application made more tractable by the presence of DTSS is library automation. Since the service already exists, the updating and maintaining of circulation and serials files can be done easily within existing capabilities. The major effort is thus spent in the design of special in-library devices to provide a quiet and dignified interface. The in-library devices connect to the central system over ordinary telephone lines. A local satellite computer drives the interface devices and provides what local editing and formatting is needed.

Educational users

A recent survey of the educational uses at Dartmouth[5] shows that approximately eighty courses (out of a total catalog of 600 courses) in fourteen departments (out of 40) used the computer in the course work. The total number of student enrollments in these courses was 5,200 (out of a total yearly student-course enrollment of some 27,000.)

The figures shown in Table VI reflect "official" computer use only and ignore casual use by students such as solving homework problems.

The extent to which computer use pervades the courses varies widely, ranging all the way from requiring the use of a canned program to series of problems for which extensive programs have to be written by the student. There are several examples of students preparing BASIC programs of more than 1000 statements for term projects.

Table VI—Use of computing services in the undergraduate curriculum

Department	Number of Courses	Student Enrollments
Biology	4	511
Chemistry	4	533
Classics	4	101
Earth Sciences	5	58
Economics	6	301
Education	3	193
Engineering Science	10	296
Geography	4	431
Government	3	98
Mathematics	16	1020
Physics	7	1000
Psychology	5	286
Romance Languages	2	108
Sociology	9	259
	—	—
Totals	82	5195
Courses on the books	600 (est.)	27000
	13 percent	19 percent

The only formal training in computing provided for students is a short exposure adjoined to the second course in freshman mathematics. Taken eventually by 85 percent of all students, this exposure provides only two lecture hours specifically devoted to computing. The remainder of the exposure is oriented toward solving problems related to the current topics in the course. Separate "programming" courses are not given. Students who need to know the more exotic characteristics about BASIC or who need to learn FØRTRAN or ALGØL are told to consult the many available manuals and texts. It is a fact that extremely simple constructions in BASIC are sufficient for perhaps 95 percent of the programs needed by students. Furthermore the student does not have to learn formatting of input data, non-intuitive grammatical or punctuation rules, or the difference between "fixed" or "floating".

The formal training program includes, besides the two lectures, four programming assignments which are coordinated with the course material.[6]

In the second term of freshman calculus, these problems are:

PIE	approximate pi by inscribed polygons
TRAP	a general trapezoid rule program
SINE	a Taylor series approximation
DIFFEQ	a simple differential equation solver

Many students elect Finite Mathematics instead of calculus as their second course. These students would see these problems:

MOD	compute A times B mod M
QUINT	root of a quintic by bisection
BDAY	probability calculation
OZ	simulated three-state Markov chain

(The first problem in each group is to "break the ice," and does not relate directly to the subject matter of the course.)

CONCLUSIONS

The Dartmouth system has had a strong influence on the curriculum. The results are almost revolutionary, although the method of their achievement was smoothly evolutionary. In four years we have seen:

- a complete change in the way homework exercises are assigned in the sciences and engineering,
- students interacting with real data in reasonably large amounts at a low cost and in a pedagogically effective manner,
- the substitution of computer simulations of physical systems in place of laboratory study of them,
- the increased use of gaming as a teaching tool, and
- new opportunities for improvement of teaching through carefully selected use of computer-assisted drill or computer-assisted instruction.

The key points here are that teachers themselves are involved directly with the educational uses and that there is little need for extensive additional systems programming.

The most important single impact of DTSS is that over 85 percent of the liberal arts student body *know* from personal experience what computing can and cannot do. Most of these students will never care what goes on inside the "black box," but they will be able to approach the subject in a rational way.

This "symbiosis" between the Dartmouth student and his computer is brought about because the design does not discriminate against the least of these users. The language BASIC, which is used for more than 90 percent of the problems, is kept simple and free from extraneous or non-obvious constructions. The command interface is likewise very simple, with the more commonly requested services being invoked by single and simple commands. (RUN invokes a compile, a load, and an execute phase, although 99 percent of all users are unaware of this breakdown.) As far as the ordinary user is concerned, the computer speaks BASIC. He never sees anything but the image of his program, properly sorted. Changes in statements are automatically sorted into their proper sequence prior to listing or running. Since only the source program in BASIC is retained, each RUN request invokes a complete compilation. However the compilers are extremely fast (between 12,000 to 75,000 statements per minute for typical small student jobs in BASIC) and thus place a relatively small marginal burden on the system. In fact, the sorting of an unsorted program with lines out of order usually requires more than half the compile time—that is, compilation is no worse than about twice as slow as sorting.

The simple fixed output format in BASIC also contributes significantly to efficiency. In a simple program, perhaps only one percent of the CPU execute time will be devoted to actual calculations; the rest will be spent converting the input data and preparing the output character stream from the numerical answers. A fixed output format is used, and can thus be implemented very efficiently.

A final important characteristic of the Dartmouth system is the quick response for small jobs. For instance, the system is designed to give fast service to initial run requests. Such a design insures as much as possible against losing the large mass of students and faculty who are the most important clients of the service.

In summary, the Dartmouth system has proved to have enormous capabilities in almost all areas of computer application in the university from small job processing for students, to larger faculty research applications, to information systems, and as a medium for CAI.

REFERENCES

1 J G KEMENY T E KURTZ
Dartmouth time-sharing
Science Vol 162 223–228 October 11 1968

2 R F HARGRAVES A STEPHENSON
Design considerations for an educational time-sharing system
Proc S J C C 1969

3 J DANVER J NEVISON
Secondary school use of the time-shared computer at Dartmouth College
Proc S J C C 1969

4 E D MEYERS JR
Project IMPRESS: Time-sharing in the social sciences
Proc S J C C 1969

5 W E SLESNICK
Educational use survey
Kiewit Computation Center 1968

6 J G KEMENY T E KURTZ
The Dartmouth time-sharing computing system
National Science Foundation Final Report April 1967

Design considerations for an educational time-sharing system

by ROBERT F. HARGRAVES, JR.

Dartmouth College
Hanover, New Hampshire

and

ANDREW G. STEPHENSON

Time-Share Corporation
Hanover, New Hampshire

INTRODUCTION

In 1963, Dartmouth College developed a time-sharing system for the GE-235 and Datanet-30 computers. This was a general-purpose multilingual computing system in which the language BASIC achieved great popularity. This system had a great impact on the Dartmouth campus, and its successes led to its adoption by GE as a commerical time-sharing system. But at Dartmouth, the success of this system led to its own demise; the demand for the use of the computer by students, faculty members, and a substantial number of outside users meant that the system always operated near its peak capacity—just under 40 users. Thus this system which initially was judged to provide a convenient and powerful computing service grew (in the view of those grown accustomed) to have certain unsatisfactory characteristics. Nonetheless, the system continued to provide good service for the functions for which it was specifically tailored—providing good edit-compile-and-go service for short BASIC programs. In fact, it was this service which drew more and more people to use the computer.

Some of the less satisfactory points of the system could not easily be remedied, however. Random access storage for retention of users' saved programs was in great demand. In order to keep a current pool of available disc storage, it became necessary to purge programs from secondary storage as they lapsed into disuse. By the time the 265 system (235 + D–30) left the campus, the purge period had dropped from 30 days to 48 hours. (It was a fact that although purged programs were always written out onto magnetic tape, no one ever felt strongly enough about the loss of a program to write the code necessary to retrieve a purged program from the tape.) In order to protect their programs, users fell to the countermeasure of conscientiously calling up all of the otherwise unused programs in their catalogs every day in order to avoid having them tagged as unused and purged from the storage. Consequently the storage situation worsened horribly.

Development of new computing languages was a difficult and hazardous task. Since the 235 hardware had no protection facilities, it was necessary for the executive system to trust the compilers not to overwrite storage assigned to other purposes. In spite of this, BASIC, ALGOL, and LISP systems were available and reasonably foolproof (from the point of view of the executive system). FORTRAN was never trustworthy enough to enable the system at Dartmouth to survive the lack of index-range checking, due to the capricious nature of students faced with an opportunity to alter the executive. A watchful interpreter for the assembly language code, TSAP, made it possible to write and debug some small subroutines, but the installation and testing of a compiler was necessarily an on-line operation on a dedicated machine.

Programs could not reasonably be allowed to access data in secondary storage. Only one user program could reside in the core storage of the 235 at one time, and the I/O wait time for a rather slow disc meant that other user programs would be faced with additional delays.

Some users felt a desire to increase the speed of computation. This was especially important for such simulation and modeling studies as were carried out by the engineers and business students in two of the professional schools at Dartmouth.

Many of the potential users of the system were unable to gain access to the system because all of the datasets and communications ports to the computer were busy. It was hoped that the number of users who could simultaneously gain service from the system could be increased.

By a cooperative arrangement with the General Electric Corporation, Dartmouth was able to replace the old 265 system with a GE-635 system. A software system had to be designed for this new machine which would satisfy the demands created by the older 265 system.

Design objectives

In addition to correcting the defects of the old 265 system, it was especially important to be able to replicate the good features of that system—namely, the ease of use of the command language, the simplicity of modifying and editing programs, and the carefree edit-compile-and-go service automatically provided for the user.

The activities of users of the 265 system were sampled over a period of a few days in order to determine the frequency of activities such as retrieving programs from secondary storage, creating programs at the teletype, listing such programs, executing user-created programs, outputting to the teletype, waiting for input from the teletype, etc. These figures based upon the number of actual users on the 265 system at the time that the measurements were made were extrapolated to 200 simultaneous users which was the design target for the 635 system The following rough generalizations were obtained:

> Every 10 seconds a log-in activity would occur.
> Every three seconds there would be an edit-compile-and-go activity initiated by the user with a RUN command.
> Every one second there would be a SAVE or OLD command requiring updating a catalog residing in the secondary storage.

A few snapshots of 265 usage resulted in the following approximate breakdown of activities: 40 percent of the users were building files by typing from a teletype; 20 percent of the users were running programs which were roadblocked for output to the teletype; 10 percent of the users were running programs which were competing for central processor time; 10 percent of the users were simply listing files on the teletype; 10 percent of the users were idle; and 10 percent of the users were performing other activities. These figures gave a very rough idea of the type of computing activity which was to be expected on the 635 system. Proposed techniques of system implementation were measured up against this imagined activity. To some extent, these activity projections are unrealistic because users expect a good deal more service from a machine such as the 635 on a per user basis. Also, the activities attributed to the user can in part be attributed to the characteristics of the 265 computing system instead; we do not really have a measure of what the user would like to be doing. Also, the character of use to which the system is put may change. If many users were to shift to highly interactive programs, the nature of the system load could change drastically.

Nonetheless, it is important to have a quantitative performance goal in mind when designing a time-sharing system and to perform *gedenken* experiments giving a 0-order approximation to the estimated performance under the most frequent types of activity. In our experiences, the majority of all design decisions were easily resolved by comparing results of such thought experiments to previously established goals. The more difficult design problems can be attacked by building simulation models and implementing these models with a computer program using a time-sharing system; the availability of a time-sharing system was a great asset in handling the harder problems of system design.

This technique was very useful in approaching the core storage memory management problem, in designing an algorithm for keeping track of the available space in secondary storage, and in designing a swap scheduling algorithm.

Thus, although the completed system was designed to provide 200 users with the same service as they received on the 265 system, the users' appetites for computing have grown to the point where it would be very difficult to find 200 who would be satisfied with the restrained sort of service they previously found so valuable.

Various classes of potential users of the computing system were recognized, and an attempt was made to predict their peculiar requirements. It was deemed important that the computing system be able to provide education in the rudiments of computer programming for the majority of undergraduates. The important function of humanizing the computer is accomplished at Dartmouth by making the computer available in the undergraduate liberal arts curriculum. 90 percent of the undergraduates avail themselves of this opportunity. It is extremely important that a simple enough command language and programming language exist so that a student is able to accomplish something in his very first session at a terminal; in this way, he is encouraged to make further use of the computer. This is not to preclude advanced features in a command language or programming language; the requirement is only that there be a small subset of these languages which may be used by the student when he is learning.

After a student has learned to use the computer, the computing system must remain a useful tool throughout the student's educational experience. Currently, there are nearly 100 undergraduate courses given at Dartmouth which make significant use of the computer. Applications in classics involve analysis of Greek and Latin text; these analysis programs rely on the existence of string-processing features in the programming language. Persons involved in the analysis of experimental data require the programming language to be able to access files in secondary storage in which this data is kept. The use of teaching programs is a blossoming field at Dartmouth, and this activity reinforces the requirement for a library in which the standard teaching programs can be stored and be easily accessible to the learning student, who is more interested in Spanish than computing. This highly interactive use burdens the swapping mechanism. Undergraduates are being introduced to the methodology of analysis of sociological data which is amassed in secondary storage. Parallel accesses to large, shared, but relatively static data bases are required. Throughout these applications, the ease of use of the computing system should be maintained in order to facilitate the introduction of computing techniques into a broad spectrum of courses.

The computer should aid faculty and graduate student research in the social sciences, the physical sciences, engineering, humanities, and the professions. These research demands often tax the computer at two extremes. On the one hand, there is the demand for raw compute power to handle such problems as occur in solid-state physics where eigenvalues must be sought for a proposed Hamiltonian describing the characteristics of a crystal. A Monte Carlo technique can be used in modeling a business activity. On the other hand, at the graduate level, both the sociological and business applications of the computer also can make use of large data bases, and the software operating system should provide the capability for randomly accessing these data.

A highly important requirement is that systems programmers should be able to perform their maintenance and system development functions in parallel with normal time-sharing activities. In order to provide as much service as possible to a broad spectrum of users, the time-sharing system should operate in its normal mode nearly around the clock. There are more benefits to this than simply relieving systems programmers from the necessity for keeping midnight hours. The computing system must be able to be developed more fully during its expected lifetime than is possible with only a few evening hours for system development work. A stagnant computing system dies; the system must be extendable in order to take on new problems. If not, the system development personnel become disinterested in the system, and it decays rapidly if there is no one left who can make even trivial changes in order to cope with the ever-occurring, newly-presented, unforeseeable circumstances. This characteristic may be unique to the Dartmouth system because there are no full-time systems programming people to develop and maintain the computing system. Undergraduates are the principal source of programming talent and ideas, and since to them this is more a labor of love than of money, they do not make strong contributions to projects in which they have lost interest. More specifically, this capability requires that editors, assembly programs, and debugging programs be made available to the systems programmers. The ability to test machine-language code without endangering the system is a must. Compilers and utility programs must not enjoy any privileged status, but must adhere to a standard set of systems interface conventions so that they may be debugged and installed without heroic surgery on the executive system. Related to the question of system development is the necessity to be able to use software developed for other operating systems for the 635 so that unnecessary effort is not wasted in duplicating such large and useful tools as macro assembly programs.

It should also be pointed out that the design of the time-sharing system was not burdened by the necessity to do standard grade recording, class scheduling, payroll, and accounting—such tasks require large amounts of paper and card handling. Of course, these tasks are necessary to the functioning of the college, but they are accomplished economically by an already existing 1401 system operating independently.

Final design

The hardware provided by General Electric was to be based upon a 635 processor. The 635 has a master mode and a slave mode. An executive program running in master mode can control all input-output facilities and supervise the use of such features as the timer register and the base address register (BAR). Memory protection and address relocation is performed on the 635 for programs running in slave mode. The BAR contains an upper bound against which all slave mode references to memory are checked, and it contains an increment which is added to the addresses of all references to memory. The existence of only one BAR means that slave programs run in only address space.

The configuration of the essential pieces of the hardware is shown in Figure 1. The drum provides a mechanism for swapping slave mode jobs in and out of core memory. The GE Datanet-30 computers provide the interface from the 635 system to telephone company datasets. Each of these D-30 computers accepts data from over 50 teletypes on a bit-by-bit basis, and transmits only complete lines of information to the I/O controller. The discs each provide 16×10^6 ASCII characters of storage which are available to the software system and provide general utility storage for users.

The architecture of the software system was influenced by three things:

The experiences with the 265 system.
The published concepts of the MULTICS system.
A realization of the limitations of the capabilities of a part-time staff of Dartmouth students and faculty members.

From the experiences on the 265 system, a pattern of use to be expected at Dartmouth was predicted. These experiences had also familiarized students with the principles of a time-sharing system. The MULTICS system provided examples of ways to generalize file structures so that a file system would be convenient to use and yet would anticipate future needs.[1] The pragmatics of getting an operating time-sharing system

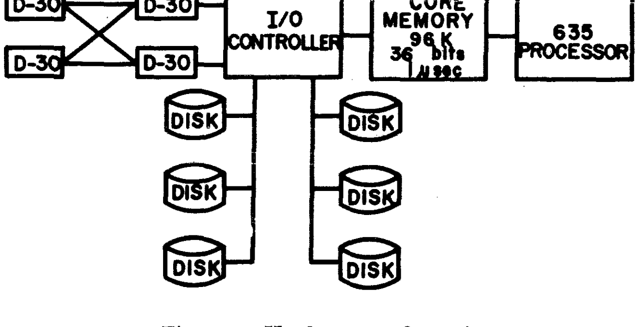


Figure 1—Hardware configuration
The D-30 computers act as communications controllers for up to 200 teletype lines. The drum is used primarily for swapping

running on a 635, using a part-time staff and having only limited debugging time available, emphasized the need for simplicity of design.

An important feature of the time-sharing system is the concept of a file. The system was designed to be file-oriented. All communications and control functions are accomplished through the medium of the file construct. The hardware of the 635 is such that the smallest convenient element for a file is a 36-bit word. In its most fundamental form, a file consists of a contiguous string of N words residing in an unspecified place with words numbered 0, 1, . . ., N-1. Programs can be allowed to alter or fetch any word in a file, accessing these words directly; the file appears to be homogeneous to the user. Since the fixed record lengths of various random access secondary storage devices for the 635 are 40 words for the discs, 64 words for the drum, and 108 words for the RACE, it becomes clear that it is necessary to isolate users' programs from such awkward record lengths. This also leaves to the executive the flexibility to select files for storage on the drum, discs, or RACE unit depending upon the amount of storage available on these devices, the history of past activity of the file, and the user's prediction of activity expected. Since all record boundaries have effectively been erased in the software, the problems of the user-written slave mode programs have been greatly simplified in this time-sharing system. The effect of this design principle is that the executive system is made responsible for all peculiarities of random access storage devices. For example, the executive takes upon itself the responsibility of handling read-alter-rewrite cycles when it is necessary to update words which form only part of a physical record. Other files in this system are simple variants of this concept. Some files, such as magnetic tape files, are naturally serial and cannot be randomly accessed. A file containing a core image of an executable 635 program can have a RUN command directed at it. This spawns a new job in the time-sharing system, and the file is then referred to as a job file.

The fundamental computational structure of this time-sharing operating system is a job. A job consists of a 635 computer program operating on data contained in one address space along with the program. Illustrated in Figure 2 are a number of files through which the job communicates. Each job in the system is an independently evolving computational process, and in principle, each can be considered to be running on its own independent computer. Jobs can only interact through files; however, this does not necessarily require the use of secondary storage facilities. The job construct has some of the flavor of the virtual

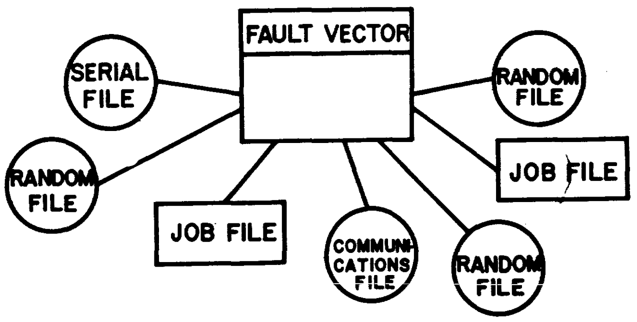


Figure 2—Job environment
A single job runs with no knowledge of the job which spawned it. It can only communicate to files. The fault vector is an exception-handling mechanism

machine techniques. Each job runs in a machine very much like a 635 computer. Words 0 to 31 of a job are reserved as a fault vector. For example, if a reference to memory is out of bounds, the address of the next instruction to be executed is stored in location 2, and control is transferred to location 3. A basic premise of this type of organization is that all unusual circumstances should be communicated to the job itself rather than causing an abort or some other activity out of the control of the running job. A running job can initiate transferral of information to or from a file by executing a coded master mode entry (MME) instruction. Each such request specifies the location of a trap to which control will be transferred upon completion of the file operation, interrupting the job at this point in order to do so. This allows a job to supervise many file activities which are taking place asynchronously. The important payoff of this type of organization is that it allows the fundamental service modules of the time-sharing system which handle file activities for many users to be structured as independently running jobs communicating via a multitude of files. Since all such modules have the uniform job structure, they can be debugged in time-sharing using the standard debugging techniques applicable to any job.

The homogeneous structure of files erases the physical record boundaries in files for programs accessing them. Therefore, the function of allocating records on secondary storage is necessarily performed by the executive system. In many operating systems, the responsibility for allocating storage for a file has been an irksome nuisance left to the programmer. The programmer is often forced to preallocate storage for a file before a program begins execution. A programmer writing in a higher-level language may very well have little idea of how much storage should be allocated. If too little is allocated, the program may be prematurely terminated because no more storage will be allocated, or the program may run inefficiently because the allocated storage has been incrementally generated according to previous specifications and fragmented into too many pieces to allow the program to run efficiently. If the programmer overestimates his storage requirements (the natural reaction), the program may run properly, but valuable preallocated storage space will remain unused. In this operating system, storage is allocated by the executive as it is needed in response to commands given by a user program to copy data into a file. The number of records allocated on a device at any one time is always a power of 2; sufficient contiguous storage space is always allocated to at least satisfy the data being appended by the current copy command. If it later occurs that additional allocations are necessary, the allocator is restricted to at least double the amount of storage already allocated for a given file. The efficiency of use of allocated secondary storage is around 70 percent. This was felt to exceed the efficiencies actually achieved by other allocation schemes in practice. The device addresses of the segments of a file which have been allocated incrementally are recorded in a file control block for the file; this allows direct random access to any word in the file with only one seek. The linked segment technique is not used since it is not suitable for randomly accessible files. Sincet he total length of the segments grows at least exponentially, the number of device addresses required to describe a file is small, and the device address list requires little core storage space. Nonetheless, if a file is felt to have been fragmented excessively, it can be reallocated contiguously in secondary storage by the simple expedient of copying it over with one copy command. This allocation scheme means that the management of free storage can be handled by a buddy technique.[2]

A communications file allows two jobs to interact directly without the use of secondary storage. A communications file has one end in each of two jobs. It is the software analog of a channel-to-channel adaptor. This structure allows job-to-job interactions using the same procedures as for more conventional files. The two ends are labeled master end and slave end. A job at the slave end of a communications file cannot easily distinguish this file from a conventional file. Since a job at the master end of a communications file can control and monitor all data transmitted on that file, a master end job can simulate a data file, thereby providing a useful debugging aid and also providing a convenient mechanism for interfacing running jobs to unexpected data structures.

Scratch files are associated only with a running job and disappear upon termination of that job. Catalog files are associated with a file structure which is part of the time-sharing operating system. The structure of cataloged files has been chosen to be a tree structure, so that algorithms which deal with this structure can be simply implemented. The file structure includes both files and catalogs. Files contain any information whatsoever and may be read or written by user programs. Catalogs contain passwords, access permission, attachment counts, and device addresses of other files or catalogs. Such sensitive data as device addresses, attachment counts, and coded dates are all critical to the proper performance of the executive system, therefore this information cannot be read or altered directly by a user job using standard file calls, but only through special catalog calls upon the executive. This flexible tree structure can be constructed or modified by user programs and was judged to be sufficient to satisfy projected needs. Cross links were not introduced into this tree structure because of the maintenance problems these links would introduce in a dynamically changing file structure. An application of the tree structure of files and catalogs to time-sharing service is shown in Figure 3. To open a file or catalog, a supra catalog must previously have been opened. Two distinct sets of accesses may be granted by the file system depending upon whether the job attempts to open the file with or without a password. For example, a job may be given read and write permission on a file if it supplies the proper password and name, and only read permission if the job does not supply a password. This two-class protection scheme supplies most of the protection needs which arise in this operating system. If a higher degree of protection is required, the catalog entry can specify that a trap program is to be run whenever a job attempts to gain access to a particular file. The responsibility of providing protection has been left up to the owner who must write a trap program which decides what access to the file should be granted.

The amount of available core storage may not meet the total demand for core storage for all of the jobs which are executing in the time-sharing operating system at one time. In this operating system, a running job may occasionally be swapped out in its entirety from core onto a drum in order to make room for other jobs. The system multiprograms those jobs which coexist in core storage; execution is alternated among jobs as input/output operations and swapping are overlapped with job execution. Since a running job is also a file, the primitive file operation performed by the executive is a transfer of data from one file to another file. Therefore previously initiated file operations can take place even if a job file has been swapped out and resides on the drum. The executive knows the instantaneous location of all job files, and if necessary it will provide buffers to continue the file transfer operation even if neither of the files involved in the transfer are located in core. This feature is especially important for low-speed input/output devices such as the teletype.

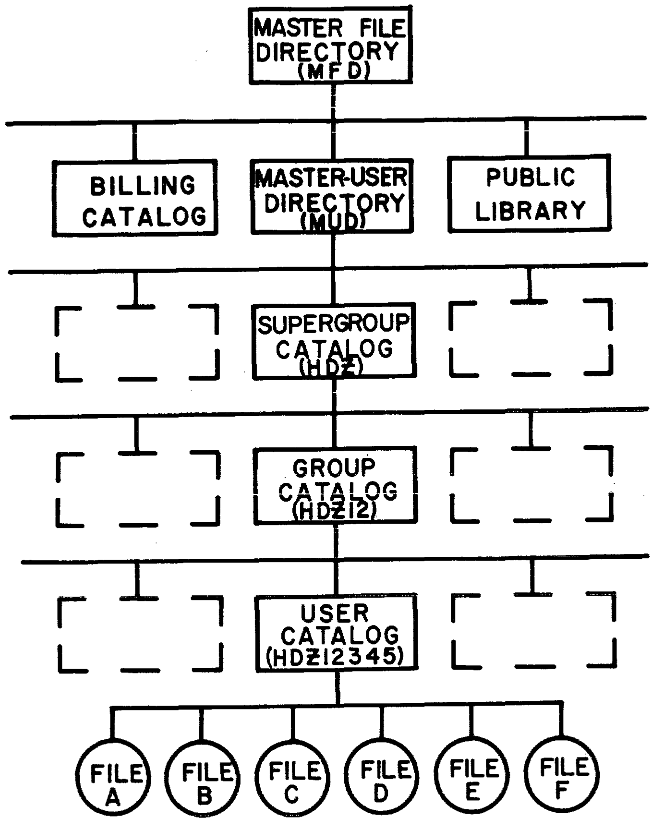


Figure 3—File structure
This application of the hierarchial tree structure shows how files are categorized at several levels

A snapshot of the job structure which exists in the time-sharing operating system is illustrated in Figure 4. The module D30INT is at the root of the job hierarchy. This module communicates over two files which are really D-30 communications control computers. These computers communicate blocks of information containing entire teletype lines to the job D30INT in the 635 computer. Information exchanges between the D-30 and the 635 computers are limited to once each half second in order to minimize the number of interruptions to the 635 and thus keep the overhead acceptable. It is the function of D30INT to block and unblock these transmissions and to provide communications for those jobs operating under the executive system which require teletype

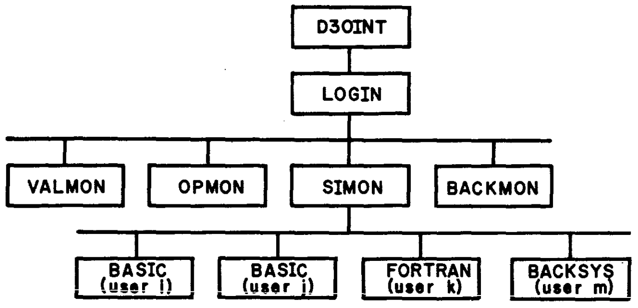


Figure 4—Job structure
This is a hierarchial tree structure. The root of the tree structure of the Dartmouth time-sharing system is D30INT

input or output. This is done by establishing a communications file between D30INT and, for example, a BASIC compiler operating as a running job in the system. Thus the job at the slave end of the communications file reads and writes an input/output device which is effectively a teletype.

When D30INT finds that a new teletype is on the system, D30INT sets up a communications file with LOGIN. The effect is that a new teletype file is presented to LOGIN. This module had the responsibility for requesting the user number and validating it, and tracing down the catalog tree structure to locate the individual user's catalog of saved programs. A portion of the tree structure with which LOGIN deals is shown in Figure 3. LOGIN then passes an end of the newly created communications file and the newly found catalog file to SIMON (simple monitor). This monitor has been especially written to provide a basic core of services for many users. SIMON provides file retrieval, editing and updating services, and calls in compilers and other service routines where necessary. At any one instant in time, most of the users on the system use only SIMON and no lower-level running job, thus reserving a good deal of system capacity for other jobs in the operating system.

The three modules D30INT, LOGIN, and SIMON have all been written to handle multiple users; they are each one job in the time-sharing system. The compilation and execution of a user's program is accomplished by SIMON by spawning a job which accomplishes this in response to a user's request. For example, if a user wishes to execute a BASIC program, a file containing the BASIC compiler is spawned in such a manner that the necessary file pre-exists when this program goes into execution. The compiler then reads the source text, compiling it directly into core and overlays the compiler with a package of run-time subroutines. The code in the compiler and the run-time package is pure procedure, but the existence of only one BAR for this machine requires that each BASIC program in execution has along with it its own copy of the run-time package. Other languages currently available under this system include FORTRAN, ALGOL, and LAFFF. LISP and ALGOL-68 efforts are under way.

Background capabilities are being introduced to allow persons with long-running jobs or jobs which require special peripheral equipment such as the card reader or line printer to initiate such service requests from a teletype. The user creates a file of information describing the background request while using a teletype in normal time-sharing under SIMON. The user types the command BACKGROUND in order to have his job control language statements checked for validity by BACKSYS. If this background request is found to be proper, it is written into a queue of background requests. A wakeup signal is then sent to the background monitor (BACKMON) which introduces these jobs into the system. This monitor operates in such a way as to attempt to optimize the mix of running jobs to achieve satisfactory performance for background jobs without undue loss of performance for foreground jobs. The file and job organizations pay dividends here: A BASIC program running under SIMON conducts teletype input/output over a communications file normally attached to D30INT and ultimately a teletype. In background operation this communications file is linked to BACKMON which supplies input from a user-specified input file, and directs output to a user-specified output file, interposing a copy of the input file for clarity.

A capacity for software simulation of other operating systems and for efficient debugging of machine language programs has been implemented by means of the SQUEEZE primitive for the executive system. This technique was very successfully used in MOLD, an earlier operating system for the 635 developed as a vehicle for the construction of further time-sharing operating systems. The name SQUEEZE is taken from the fact that the base address register (BAR), which limits the range of allowable memory references, is set to a smaller range (squeezed) by this call. Additionally, all actions of an executing program which would normally cause some executive response, such as execution of a MME (master mode entry) instruction, divide-check, etc., are brought to the attention of the program which issued the SQUEEZE by resetting the BAR to its unsqueezed value and saving the program status appropriately. The effect of this simple technique is that a program may be embedded in an outer program, and that outer program can be written to interpret MME's, divide-checks, etc., of the inner program in any way deemed fit. An outer program called DDT has such a capability, and allows patching, dumping, and breakpointing capabilities for debugging any of the jobs in the system.

If the SQUEEZE technique were useful only for debugging programs, it would have been well worth the effort for this cuts down checkout times for assembly-language jobs by a factor of ten over the load-run-die-dump technique. However, this technique also reaps benefits in providing the capability to write outer programs which can simulate the action of other operating systems for the 635. For example, the GECOS operating system is simulated using this technique. The macro assembly program, GMAP, developed to run under GECOS, runs normally under this simulator until a MME or other cause for executive action is encountered. The simulating program determines the action which GMAP expects GECOS to perform, and it initiates a corresponding action by making an appropriate call in the time-sharing operating system. This technique has meant that many useful programs operating under other 635 operating systems have been taken over directly without modification and function usefully in the current time-sharing system.

ACKNOWLEDGMENTS

The work described in this paper was performed by a group of persons under the direction of John G. Kemeny, and the authors are pleased to have been privileged to work with these people. Thomas E. Kurtz, Stephen J. Garland, John S. McGeachie, and Gordon Bull made significant contributions. Sidney W. Marshall and Neal F. Weidenhofer were Dartmouth graduate students who were essential to the success of the project. The Dartmouth undergraduates who were responsible for major aspects of the system were David Magill, Peter Nielsen, Steven Hobbs, Gregory Dobbs, Anthony Dwyer, Paul Velleman, and Andrew Behrens. Ronald Foottit, William Forfar, and Rajendra Kanodia accelerated progress during six-month visits.

REFERENCES

1 F J CORBATO V A VYSSOTSKY
Introduction and overview of the multics system
Proc F J C C 1965

2 K C KNOWLTON
C A C M Vol 8 1965 623–625

A flexible user validation language for time-sharing systems

by JOHN S. Mc GEACHIE

Mandate Systems Inc.
New York, New York

INTRODUCTION

Dartmouth College has recently developed a general purpose time-sharing system for its General Electric 635 computer, capable of supporting approximately 150 simultaneous users. The system provides computing services to some 3000 faculty and students on campus, as well as to more than 6000 teachers and students at over 30 colleges and secondary schools scattered throughout New England. At any given time during the school year, therefore, there are close to 10,000 potential users, of which some 4000 use the time-sharing system every month.

Each of these potential users (henceforth referred to as "subscribers") may access the system to create, store and retrieve his own programs, and to make demands upon the computer's time for the processing of these programs. As users are billed for their use of the computer and associated storage devices, it is highly desirable that each user uniquely identify himself to the computer in order to prevent masquerading, i.e., Smith may not pretend that he is Jones so as to access Jones' programs at Jones' expense. In addition, it is conceivable that a given user might captivate sufficient system resources to significantly degrade the service that the system provides to the user community as a whole. This is because a time-sharing system operates by giving a small portion of the total system to each user for a short period of time, and any user who absorbs too much of the system in time, space, or I/O capacity may cause a disproportionate loss of service to the remaining users.

There are therefore two clearly distinguishable functions which must be performed when someone attempts to access the time-sharing system. First, is he allowed to use the system at all? Second, if he is allowed access, then what resources are available to him? The twofold task of defining adequate security controls and reasonable resource limitations for each of 10,000 users turns out to be quite complex.

The process of providing the computer with the required information about each subscriber is called the *validation* process, and the purpose of this paper is to relate some of the approaches taken to the problem of validating such a large number of subscribers. A number of the techniques developed appear to be applicable to time-sharing systems other than the one at Dartmouth College. The process of matching the stored computer data with an incoming user is called the *login* process; it will not be covered in this paper.

Before proceeding it will be useful to describe some additional factors relevant to the validation process. The sheer volume of information is considerable, but the job is further complicated because it is not "one-shot", i.e., it is not possible to tell the computer lots of things about ten thousand people and then forget about it. The academic environment in which the Dartmouth Time-Sharing system (DTSS) operates has an interesting property: every June, large numbers of subscribers graduate, and every September there are roughly 3500 new ones in the colleges and high schools that make regular use of the system. In addition, every term a few courses are taught whose students require some special privilege in order to access a resource not normally available. Examples of system resources are: core memory, central processor time, secondary storage (drum, disc, etc.), and all peripherals. Certain privileged system functions are also grouped in the same category, such as the ability to change subscriber privileges, to access system accounting data, etc.

Of the many requirements that the validation program (used to feed subscriber data into the computer system) would have to meet, two were of paramount importance. First, the large numbers involved made it imperative to permit manipulation of sizable blocks of users as single entities. Second, the program command language would have to be simple enough to enable secretarial and administrative personnel to handle all of the validation procedures. The latter requirement was partly because the operators and programmers were too busy with other essential activities, and partly because that type of function seemed to be an obviously administrative one.

I shall now attempt to describe how these objectives were achieved. The fundamental identification of each subscriber within the time-sharing system is his *user number*. This is an eight character identification sequence composed of two parts: the *group* (first five characters), and the number within the group (last three characters, which are numeric). A group is largely an administrative entity; for example, most of the subscribers from any given high school would belong to the same group, as would most of those from any one corporation. Thus user numbers HDH31000 through HDH31999 would all belong to group HDH31. Associated with each user number there may be an arbitrary sequence of up to eight letters or digits which the user must apply to the computer when attempting to access the system: this is called a *password*, and provides an additional level of security. For example, the password for most students at Dartmouth College consists of the first three letters of their last name. Furthermore, most of the terminals connected to the system are teletypewriters (leased from the telephone company), each of which has an identification sequence (called an answer-back drum), which may be interrogated by the computer. It is therefore possible to determine from which device a given user is calling, and to refuse him service if he is calling from a device for which he is not authorized.

The security provisions are thus controlled by three factors: the user number, its password, and an answer-back drum. Whenever a user attempts to access the system, he supplies the first two, and the latter is automatically transmitted to the computer upon receipt of a special signal. An additional control is provided by the time of day. It was considered useful to be able to restrict certain users to specific periods during the day, and consequently each user number has associated with it a *legal log-on time*, defining those time periods during which he may access the time-sharing system.

Given that a user can access the system, i.e., he supplies the correct user number and password, and is using an "authorized" terminal during a "legal" time of day, the computer now determines the quantity and type of system resources that he may use. This is done by means of various parameters associated with each subscriber's user number. Experience with previous time-sharing systems at Dartmouth College has shown that the most critical resource is disc storage, used to save user programs between sessions with the computer. Two parameters control disc usage: one, the *catalog storage* allotment, determines the total amount of storage which a user may occupy with his saved programs; the other determines the total amount of temporary storage that he may use while actively accessing the system; it is called the *scratch storage* allotment, and everything in scratch storage is lost when the user signs off the system. Another parameter determines the amount of continuous central processor *running time* that a subscriber may use.

Finally, each user number has a set of special *permissions* associated with it. These permissions are of the on/off type, and control a wide variety of miscellaneous functions such as: the ability to use the Background system (for very large or long running jobs); the ability to use the Operations Monitor (control of system functions such as shut-down, accounting, statistics, etc.); the ability to use the printer; and so on.

The totality of the data required for each subscriber is called his *validation record*. All the validation records for a given group are stored in a *validation file* on secondary storage. The Dartmouth system has a tree-structured file system, in which the validation data files are accessible much like ordinary user files or programs, except that they are better protected.

In order to cut down on both the amount of typing and the disc space required to create and store a validation record, a user classification scheme was developed which takes advantage of the fact that there are significant subsets of the user population who have approximately similar requirements for storage space, running time, log-on times, permissions, etc. Each subscriber is assigned a *user code* which becomes an integral part of his validation record; this code is completely independent of the group to which the subscriber belongs, and separates users into categories having similar requirements. For example, all high school students are assigned one user code while all high school teachers are assigned another, thus placing them in different categories; but all the students and teachers from the same high school belong to the same group (and thus their user numbers all have the same first five characters).

Given that there are large subsets of users with similar requirements, it would be wasteful to duplicate a lot of information in each validation record. Accordingly, validation records have variable lengths. Any information not present in the validation record is obtained at log-on time from a special data file called a *default file*. This file is indexed via the user code, and contains one fixed length entry for each possible user code. These entries contain data relating to system resources and special permissions; they are not concerned with passwords or answer-back drum sequences.

A further simplification is achieved by noting that most of the users belonging to a given group will tend to use the same terminals, i.e., most company employees will use their company's terminals and most high school students will use their high school's terminals. In most cases, therefore, the answer-back drum sequences merely need to be specified as common to the whole group and not duplicated for each individual subscriber, although there are, of course, exceptions.

A special purpose program together with an associated command language were designed and implemented to process all the relevant subscriber data. The program converts this data into a format which can be referenced by the login program whenever someone attempts to access the time-sharing system.

The validation program contains all the mechanisms necessary for the creation of groups, user numbers, and default entries; for the specification of passwords, legal log-on times, and answer-back drum sequences (either common to a group or pertaining to a particular user); and for the allocation of system resources and associated limitations. Some additional features were installed to allow the convenient validation of large numbers of subscribers, such as the ability to accept input from a validation source file. This file can be created under normal time-sharing conditions, checked, corrected, and then given to the validation program, which will treat it exactly as if the information were being typed at a terminal.

At this point we discovered we had a powerful and flexible tool at our disposal. The number of different combinations and special arrangements possible was virtually unlimited. One of the most valuable consequences of the default file is the ability to modify the resources available to several thousand users with only a few commands. In particular, as more experience is gained about the behavior of the time-sharing system under various conditions, it will be possible to quickly and simply vary certain factors in the load that any given category of user may impose on the system. The structure of the validation records is such that, while most users will be assigned default parameters based on their user code, exceptions can be quickly made when required.

Structure of the validation command language

The general appearance and structure of the validation program commands is best described with the aid of some examples. Consider the command sequence

CG HDC92;*CS* 1000;*AU* 739, JSS;*PR* OPR,BAK;
UC 63

The italicized portions are command verbs; the other strings are arguments. The line terminates with a carriage return. The first command, *CG*, indicates that we are going to modify group HDC92. In order to avoid the necessity of remembering which groups exist and which do not, the group will be created if it does not exist; the command *AG* (Add a Group) does exist, however. The next command, *CS*, indicates a modification to the group catalog storage limit: this is the maximum saved storage allowed for all the users in the group, summed together. The units are in thousands of GE-635 words (36 bits), and the command therefore indicates a limit of one million words for the group catalog storage.

The next command, *AU*, adds number 739 to the group, yielding user number HDC92739 with password JSS. An error message will be printed out if this number has previously been defined for this group. The command *PR* will specify certain special permissions, in this case Operator (may use Operations Monitor) and Background (may use Background system). The last command (*UC*) specifies the user default code. The carriage return will cause the validation program to create a validation record according to the data supplied. Note that any items missing (in particular, catalog and scratch storage limits, running time and legal log-on times for this user) will be supplied from default file entry number 63 when the user attempts to log on the system.

The commands *CG* and *AU* are termed *major* commands, while *CS*, *PR* and *UC* are *minor* commands. All minor commands modify the immediately preceding major command. The ordering of minor commands is irrelevant, as is the presence or absence of spaces between commands and arguments.

Another example is

CG HD514;*AU* 501,ADE;506,FOO;295,
NONSENSE;*LT*12-23;*UC* 5

In this sequence group HD514 is to be modified; if created, the standard group catalog storage limit of 250,000 words will be supplied. User numbers 501, 506 and 295 will be added to group HD514, with passwords ADE, FOO, NONSENSE, respectively. The command *LT* sets the legal log-on times to noon to midnight. The user code is set at 5. Note that the minor commands apply to *all* the user numbers appearing as arguments to the previous *AU* command.

Still another example is

> *CG* HDH31
>
> *AU* 101-562;695,MNS; *CS* 4; *SS* 20; *RT* 8; *LT*12-23; *PR* BAK; *UC* 9

These command lines will create user numbers HDH-31101 through HDH31562, with no passwords, and user number HDH31695 with password MNS. Each of these user numbers will be allowed 4000 words of catalog storage (*CS*), 20,000 words of scratch storage (*SS*), eight seconds of running time (*RT*), p.m. access (*LT*), ability to use Background (*PR* BAK), and a user code of nine.

Default file comands have the form

> *AD* 6; *RT* 16; *CS* 2; *SS* 20; *LT* 6-20; *PR* BAK

This example will create a default file entry for user code six with 16 seconds running time (*RT*), 2000 words catalog storage (*CS*), 20,000 words scratch storage (*SS*), log-on times of 6 am to 8 pm, and ability to use the Background system. Any user with a code of six whose validation record omits a system resource parameter will be assigned the appropriate value from the above default entry. If, for example, a category six user has no running time explicitly specified, he will be allowed 16 seconds.

Answer-back drum sequences are created by commands similar to the following:

> *CG* HDC55; *AA* 646-2643; *AU* 601; *AA* 646-3259

The effect of the first two commands is to allow all users belonging to group HDC55 to log-on from the terminal whose answer-back drum is 646-2643; the second two commands specify that user HDC55601 may log-in from an additional terminal (646-3259). The answer-back codes as specified are not quite accurate, since they contain a number of leading and trailing non-printing format characters, which are difficult to represent here. It is possible for a user's validation record to specify that his answer-back drum is not to be checked, even though his group may have some common answer-back entries.

Finally, the example

> *PG* HDC10; *PU* 0-999

may result in the following printout

```
      GROUP HDC10, CAT MAX 244
UN 087-101  PW SBNK   UC 57   CS 500
UN 259-259  PW RMJ    UC 62   CS 510 SS 1000
```

This printout gives the validation records for all valid users in group HDC10, in this case HDC10087 through HDC10101 and HDC10259. Only those parameters explicitly specified in the validation record are printed out.

Some remarks on the internal structure of the validation program

As is often the case, a cleanly designed, well-formed syntax greatly simplifies the problems of implementation. The validation program is no exception. Transition diagrams were drawn for the whole language, and a good many of these were encoded into the program. Whenever possible, tables have been used to drive the program, including, for example, a table to control the printing and formatting of validation entries.

The main program loop is called the "diagrammer," and uses a four-entry table whose components are as follows:

Word 0	ASCII representation of major command
Word 1	Address of subroutine to decode major arguments
Word 2	Address of subroutine to decode minor command lists
Word 3	Address of subroutine to process major command

The minor command list decoder in turn uses several three-entry tables to control the processing of minor commands and their arguments: the format of these tables is similar to the major table described above, namely:

Word 0	ASCII representation of minor command
Word 1	Address of subroutine to decode minor arguments
Word 2	Address of subroutine to process minor command

An immediate consequence of this scheme is that an automatic sequencing mechanism is provided to call the appropriate subroutines in order. The tables also make the debugging very much easier.

The program is quite insensitive to the order in which minor commands are typed, although for major commands it requires that a group command precede the user commands (*AU*, *PU*, *CU*, *DU*) The entire language was designed and coded in approximately two months, and is currently being used to validate users for the new system. The program is well protected, accessible only from two control terminals and by a very small set of user numbers. One of the advantages of the ability to take input from a validation source file is that this file may be prepared from any terminal, thus minimizing the time during which a control terminal must be tied up.

An important goal of the implementation of the validation and login programs was to keep the size of the validation data files to an absolute minimum. There are a variety of different entries possible within a validation file, most of the variations being in the interest of conserving space. The detailed format of the validation records is given in Figures 1-6. Each entry type is designated by a code in bits 27-29 of the zeroth word of the entry. The various entries can be broken down into separate categories, of which the first is the "short" entry, used for validation records that specify no answer-back drum checking and no explicit system resource parameters. Figure 1 shows the format of single user entries with and without passwords. All two letter abbreviations refer to quantities used in the command language.

Figure 2 shows the validation records containing information for a block of consecutive user numbers.

Figures 3 and 4 represent "long" entries, used whenever special permissions, accesses or answer-back drum entries are required. Note that the upper and lower limits of a consecutive user number sequence may be equal.

Figure 5 shows a special entry which is always the first validation record in a file. It contains the information for a given user group, including the common answer-back drum sequences (if any).

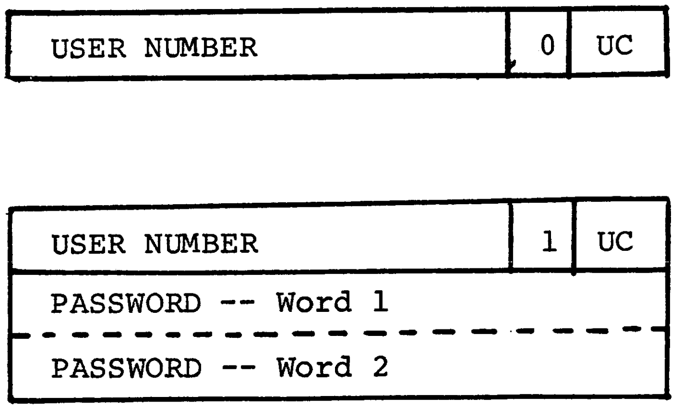


Figure 1—Single user number short entries

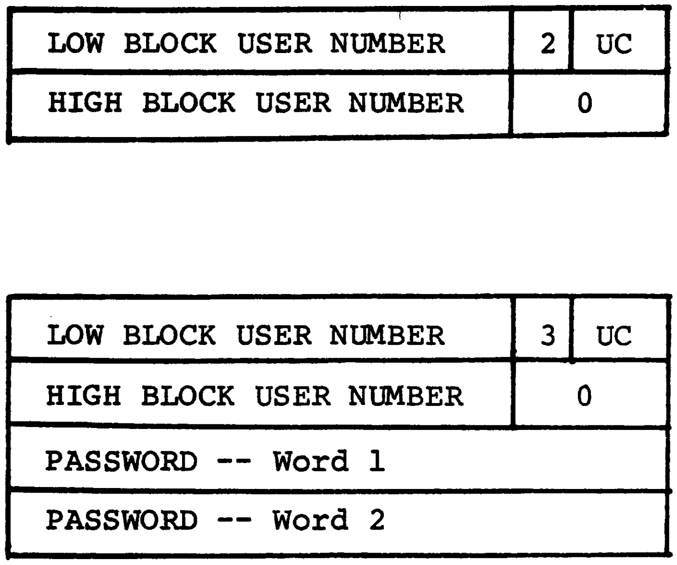


Figure 2—Consecutive user number short entries

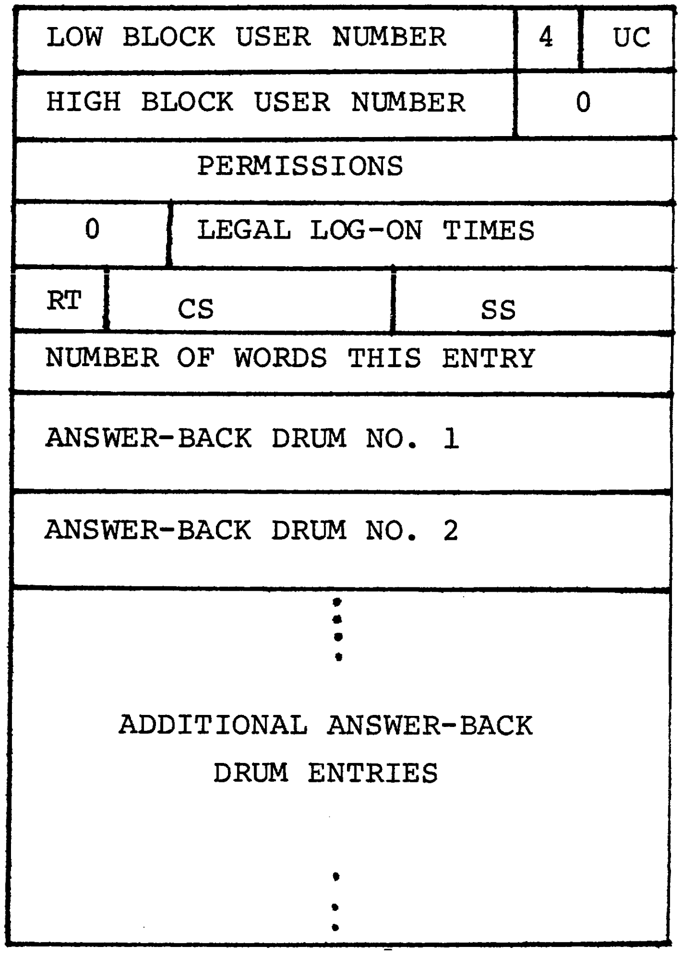


Figure 3—Long entry without password

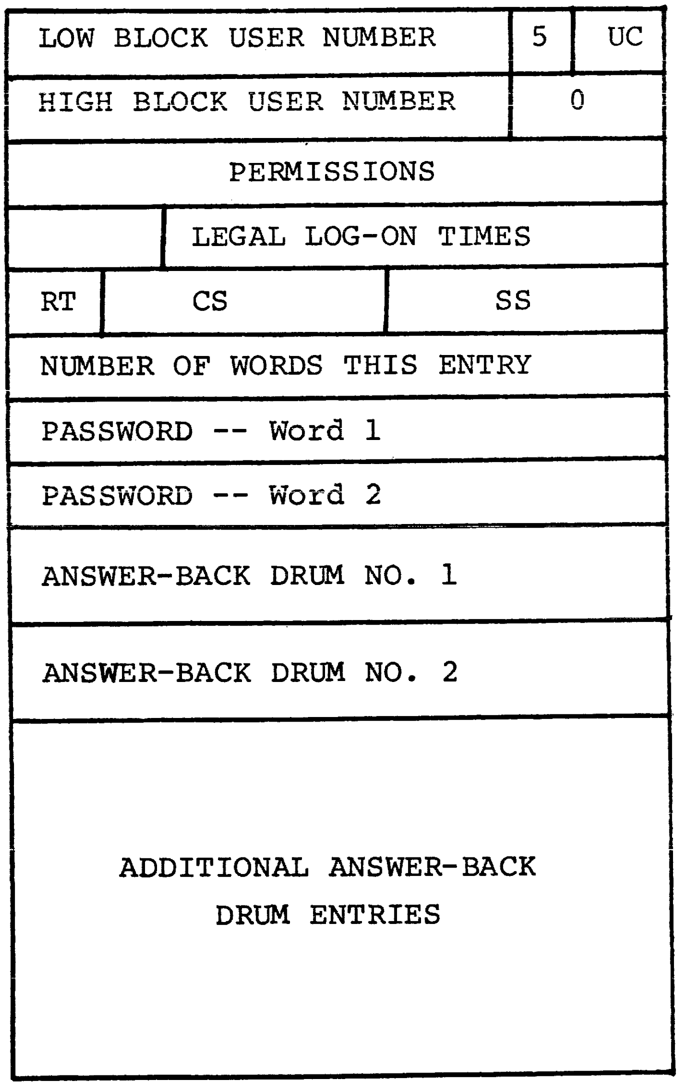


Figure 4—Long entry with password

Figure 6 shows a special entry used to terminate a validation file; it is present in every such file.

The permissible range of values for the catalog and scratch storage parameters is from one to approximately 65 million words. The presence of a zero is a signal to use the default file parameter, and the presence of all ones is a signal to use the absolute system maximum of approximately 68 billion words. It was felt that 65 million words was sufficiently large to make any gradations between that and the absolute maximum meaningless. The running time parameter is stored in the form of the base two logarithm of the value supplied to the validation program, and ranges from one second to approximately four hours.

Table I lists the validation commands together with a brief description of their functions.

Table I—List of validation commands

AG	—Add a group
DG	—Delete a group
CG	—Change a group
PG	—Print a group
AU	—Add a user number
DU	—Delete a user number
CU	—Change a user number
PU	—Print a user number
AD	—Add a default file entry
DD	—Delete a default file entry
CD	—Change a default file entry
PD	—Print a default file entry
VI	—Accept input from a validation source file
VO	—Direct output from print commands to a data file
EX	—Exit from the validation program
UC	—Specify a user code
PR	—Specify permissions
LT	—Specify legal log-on times
RT	—Specify maximum running time
CS	—Specify maximum catalogued file storage
SS	—Specify maximum scratch file storage
AA	—Add an answer-back drum sequence
DA	—Delete an answer-back drum sequence

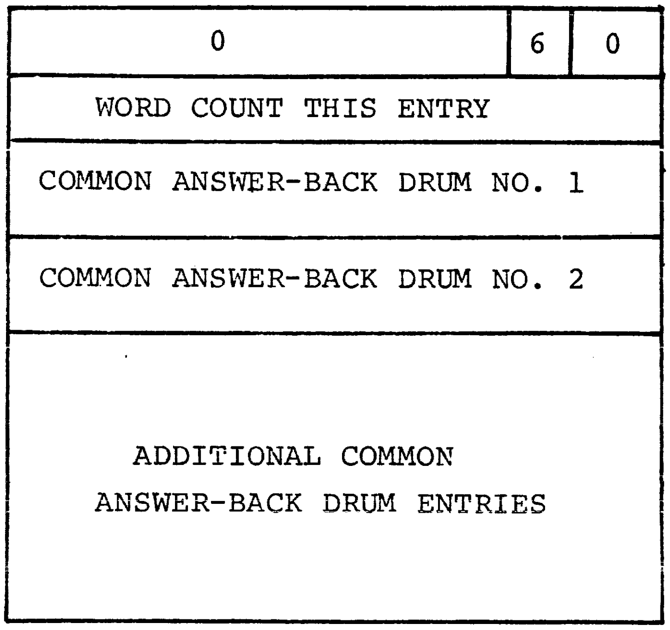

0
6
0
WORD COUNT THIS ENTRY
COMMON ANSWER-BACK DRUM NO. 1
COMMON ANSWER-BACK DRUM NO. 2
ADDITIONAL COMMON ANSWER-BACK DRUM ENTRIES

Figure 5—Group validation record

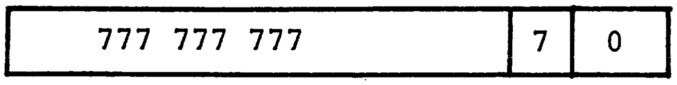


Figure 6—Validation file terminator

REFERENCES

1 J G KEMENY T E KURTZ
The Dartmouth time-sharing computing system
National Science Foundation Final Report April 1967

2 T E KURTZ
The many roles of computing on the campus
Proc S J C C 1969

3 J DANVER J NEVISON
Secondary school use of the time-shared computer at Dartmouth College
Proc S J C C 1969

Project IMPRESS: Time-sharing in the social sciences*

by EDMUND D. MEYERS, JR.

Dartmouth College
Hanover, New Hampshire

INTRODUCTION

Assuming that one has access to a large, general-purpose time-sharing system, social scientists would like to have several computational resources. First, large quantities of data should be easily accessible on-line. Second, it should be possible to retrieve and statistically analyze these data without being a skilled programmer. Third, undergraduates should be able to become involved with data analysis, and the involvement should be accomplished with relative simplicity. Finally, faculty should be able to carry out their empirical research—no matter how complex—with equal simplicity.

One of the major functions of Project IMPRESS is to design and implement such a social science package on the Dartmouth time-sharing system. The IMPRESS system may be viewed as twenty or thirty interrelated computer programs in BASIC and time-sharing Fortran, yet it is more than that. The system is an interface between the data and the social scientist; that is, the user may identify data files and variables by their customary labels, and the system will translate this information into file calls, etc. Furthermore, the system is designed to fetch and reduce raw data as efficiently as possible in order to minimize the burden placed on the time-sharing system.

It is often frustrating to examine a new system, since the reader is not always clear as to what is day-dreaming, what has been designed, and what has been implemented and debugged. With respect to the present Project, a prototype system has been fully operational for some time. The actual IMPRESS system has been designed, implemented in part, and is in stages of debugging and expansion.

A sample problem

Perhaps the discussion of the task and the solution would be more clear—especially to those lacking experience in empirical data analysis in the social sciences—if we start by looking at a sample problem.

As a sociologist, I have certain interests concerning the nation's population. So, for a typical data file, let us consider the 1-in-10,000 sample of the 1960 U. S. Censuses of Population and Housing. This file is available on IBM cards from the Census Bureau; each card represents a single person, and there are 17,939 cards in the sample. In round numbers, we are considering eighty pieces of information for each of 18,000 persons *or* 1,440,000 numeric values to be stored, retrieved, and manipulated. With respect to most of these data, the numeric values represent nominal codes; for instance, zero might represent males, and unity would represent females.

Given data of this sort, what sort of analysis is desired? In simplistic terms and merely for the sake of an example, one might be interested in the associations among income, occupation, and education. In addition, it should be rather obvious that either sex or race or both might seriously affect the associations. For example, given certain levels of education, do blacks obtain the same types of jobs and/or income as do whites? In order to determine whether blacks are "over-educated" for the jobs they obtain, one would want to construct a contingency table which included the relevant variables.

The two-stage system

Now, let us compound the problem by posing the empirical question to students in an introductory sociology course. What will happen when one hundred students attempt to use the file? In order to be certain

* Project IMPRESS (*I*nterdisciplinary *M*achine *P*rocessing for *R*esearch and *E*ducation in the *S*ocial *S*ciences) is supported by a grant from the Carnegie Corporation.

that the load these students add to the time-sharing system is not fatal, it is mandatory that data retrieval and analysis be accomplished by means of a two-stage system.

With very little thought, the individual can recognize that he is interested in an examination of five variables, as follows:

1. Income	(7 categories)
2. Occupation	(10 categories)
3. Education	(5 categories)
4. Sex	(2 categories)
5. Race	(2 categories)

Consider the complete cross-classification of all five variables; this would produce a table containing

$$7 \times 10 \times 5 \times 2 \times 2 = 1{,}400 \text{ cells.}$$

By reading the raw data once and reducing this very large file to the quite managable one containing 1400 values, it is rather easy to write programs to obtain two-variable or three-variable tables from the five-variable table. Specifically, it is accessing the storage devices which is highly time-consuming and a potential burden to time-sharing. Notice that once the 1400-cell tabulation is obtained, the individual may be occupied for thirty minutes in the analysis of the smaller tables which are simple reductions from the large one. He is doing exactly what he wants to be doing—examining the various relationships among five variables, taken two, three, or even four at a time—while imposing a lighter-than-average load on the system.* Notice further that, should this much accessing of storage be a burden on time-sharing, the instructor can perform the data reduction and then send his multitude of students off to teletypes to work with the small 1400-cell tabulation.**

The imposition of the two-stage system upon the student or the researcher appears at first glance to be a limitation. Actually, this is not the case. Typically, given a clear statement of the substantive question, the social scientist can formulate a brief list of variables

* Logically, there are 325 possible tabulations which could be obtained from the given 1400-cell table. Of course, there may be only a dozen or so which are meaningful or of interest.

** With a modest amount of ingenuity, this approach need not be a boring exercise. Suppose the instructor poses a substantive question and then provides a reduced table consisting both of relevant variables and of a few which appear to be interesting but are actually irrelevant to the original question. The challenge is still inherent in the task, and each student has the experience of attempting to answer a substantive question by examination of empirical data.

which he suspects are relevant to the question. If examination of one or more of the sub-tabulations obtained in the second portion of the two-stage system indicates the desirability of including additional or different variables, one can always return to the beginning and start again.

Technical details

Having treated our project in a rather superficial manner in order to give some indication of the nature of the task, I would like now to examine some of the technical details.

First, the storage problems which face social scientists should be clarified. If the Census file of 1,440,000 numeric values is to be considered as average, then roughly one hundred data files of this magnitude should be available on-line in the time-sharing system.* (1) It is necessary to obtain a storage device which is large enough to hold this much data and fast enough to retrieve it promptly, and it is necessary to load the data into the storage device. This is not a major problem, since several acceptable devices are on the market; loading the unit need be done only once, and no one is especially concerned about the execution time of "once only" tasks. (2) Reading from the storage device during time-sharing is a serious concern. The two-stage system is one means of reducing this problem; efficient means of storing and retrieving data in order to obtain further reductions are discussed below. (3) The speed of the output device (a teletype operating at ten characters per second) appears to be a serious limitation to those social scientists with batch-processing backgrounds; when one is accustomed to generating masses of paper, the slowness of the teletype is irritating. On the other hand, as the scholar learns to ask for only those values in which he is interested and when—due to the time-sharing environment—he has the option of asking for other values immediately if he should change his mind, the output problems tend to become negligible.**

Second, the development of a time-sharing system for social scientists is particularly dependent upon (1) simple, powerful, and flexible file-handling capabilities

* There is considerable variation in the size of data files in the social sciences. For example, in the present holdings in the Sociology data library at Dartmouth, there are forty-five files. Sample sizes range from N = 115 to N = 20,345, with the mean N = 2,190. The number of cards per observation (which may be misleading, since the number of columns used varies extensively) ranges from 1 to 20, with a mean of 5.9.

** Furthermore, if extremely large quantities of output are unavoidable, as in a large stepwise regression procedure, it is always possible to by-pass the teletype and obtain the output via the on-line printer at 1200 lines per minute.

and (2) the capacity to chain from one program to another. Both capabilities are available in the Dartmouth-written time-sharing executive for the G. E. 635. As one should be able to see already, the nature of the processing tasks require either large amounts of memory or the capability of segmenting tasks by chaining from one routine to another.

Now we can consider the details of the total process in terms of three distinct steps: (1) data storage, (2) data retrieval and reduction, and (3) data analysis

Data storage

Looking back to the card form in which we obtained the sample data from the Census Bureau, it is observed that each card column contains a single punch. Since, including "blank," there are thirteen possibilities in any card column, it is ridiculous to waste a 36-bit machine word on each individual datum. Four binary bits are more than sufficient to handle thirteen possibilities; so the 4-bit grouping is our basic unit of analysis. The grouping of four binary bits is equivalent to a semi-byte or a "nibble." There are nine nibbles to a machine word; by packing data in this form, we increase the effective capacity of the storage unit by a factor of nine.

More by chance than by design, there are certain advantages to working with the nibble. Thus far in the discussion, we have limited consideration to nominal data which can be contained in a single card column or in a single nibble. Of course, we want our system to be able to treat annual income, population, I.Q. scores, and other continuous or multi-column variables. Using a one-for-one translation from card columns to nibbles, a nice pattern emerges *after the single nibble*. The left-most or high-order bit is never used to record the numeric value. Therefore, this bit may be used as a sign bit.

Number of Decimal Digits	Largest Decimal Value	Binary Equivalent (*grouped by: nibbles*)
2	99	0110 0011
3	999	0011 1110 0111
4	9,999	0010 0111 0000 1111
5	99,999	0001 1000 0110 1001 1111
6	999,999	0000 1111 0100 0010 0011 1111

As a practical limit, it was decided that a signed six-digit field will represent a maximum; for most of the work done in the social sciences, six digits is more than sufficient accuracy. Furthermore, by storing the assumed location of the decimal point as part of the header record for the data file, one obtains the final degree of flexibility required in this scheme of data storage. In sum, the use of the nibble as the basic unit permits the storage of either single-column nominal data (with $2^4 = 16$ possible categories) or multiple-column continuous data consisting of signed, rational values up to six digits in width.

Another advantage of this scheme emerged by chance and not by design. The Dartmouth time-sharing system automatically creates file buffers of 0.5 K. Since we effectively increase everything by a factor of nine, these buffers appear to us to contain 4,500 nibbles each. It turns out that the first six counting numbers are each integer divisors of 4,500; so—in unpacking the data—we are never left in the awkward position of having one portion of a multiple-nibble value in core with the remainder about to be read into the buffer with the next access of the storage unit.

One of the questions frequently raised by my colleagues concerns the local "political" decision of which data are to be stored on-line in the storage unit. The present plan is to utilize approximately two-thirds of the available storage for "classic" data files obtained from the various data archives. The balance of available storage will be allotted to individual research projects as needs arise. By using magnetic tape back-up for infrequently used data files, it should be possible to service the large majority of requirements without delay. Except during the very busiest time-sharing hours, one should be able to retrieve even the most obscure file in less than one hour.

One of the key issues which had to be faced was the format in which data are stored. There are two possibilities: bucket format and vat format, where one is simply the matrix transpose of the other. Bucket format is typical of most counter-sorter operations; the record (the IBM card) represents a respondent. Vat format is less widely used; there, the record represents a variable. Consider the advantages of each format type in terms of social science needs in a time-shared environment.

For many research problems and almost all pedagogical work, vat format is the obvious choice. Since each record represents all respondents for a single variable, every numeric code transmitted from the memory unit to the C.P.U. is utilized in the construction of the cross-classification table. Since there is no "excess baggage," it is extremely efficient. In the example with Census data, given the file structure and the buffer size imposed by the system, the five-variable table can be constructed via twenty accesses of the storage unit if vat format is utilized; to perform the same operation in bucket format would require 320 accesses of that unit. Since most of the work to be accomplished can be done in this manner, it is obvious that vat format is desirable.

After becoming involved with all of this, we discovered some serious limits to the use of vat format. At the moment, there is a system-imposed limit of eight files which may be opened by any user at any point in time. For a certain class of problems, this is unreasonably small. Consider the researcher working with dichotomies; he would be limited to an eight-variable tabulation *or* $2^8 = 256$ cells. If it were not for the eight-file buffer limit, we could easily handle a twelve-variable tabulation or $2^{12} = 4{,}096$ cells. Indeed, techniques are described below which permit even more variables to be included.

Because of the system-imposed limit on the number of open files and because of the need to have available numerous variables simultaneously, it appears that data must be stored in bucket format as well as in vat format. With many mass storage devices (especially the ones which are extensively electromechanical in design), file security requires duplication of all records. The obvious solution is to have one of the copies be in bucket format and the other in vat. Of course, there is a cost to be paid for this; should a read failure occur in one format, it will be time-consuming to perform the matrix transpose from the other format before the task can be accomplished.*

In addition to storage of raw data, it is necessary to maintain header files containing relevant parameters for each variable in every study on the system. Before going to the details of the header files, it should be mentioned that a problem was encountered in efficiently relating header information to the raw data. Specifically, we found that we could not customarily use the data in the form in which we obtained them. Of course, it is important to maintain the data in the rawest possible state; however, it was necessary to recode all nominal data such that the single-nibble codes started with zero and continued in sequence. Obviously, this adds significantly to the burden of data preparation: again, since this is a "once only" task for each data file, it appears to be the most efficient approach.

Within the present time-sharing system, there are three types of files: (1) pure numeric files, (2) pure string files, and (3) "teletype" files. The numeric and string files are relatively strict in structure, but they permit random access operations within files. A teletype file is quite flexible, may contain a mix of numeric and string data, and must be treated serially. For most of the work, the best approach is to look at a numeric file and a string file in tandem. Category names, such as "blue collar" and "white collar" are stored in string files, and code values are stored in the corresponding numeric files. As simple points of information, the reduced data file (the five-variable table in the example) is stored in a teletype file, for it is necessary to include certain string variables along with the numeric cell frequencies. The raw data are stored in pure string files; while the data are logically numeric in nature, string files are used in order to accomplish the packing and unpacking of nine nibbles in each machine word.

* The alternative solution to the task (double storage of bucket files and double storage of vat files) is even more costly, since this would cut in half the amount of data which can be maintained on-line.

Looking at all of the files in logical order, the first is a file containing the names of studies available on the system. While the user may identify the study of interest to him by name, the system can obtain the code name of the header file for that study from this first "table of contents" file. For each study, there are files containing the names and locations of variables contained in the study; in addition, there are header files for each of these variables. The important feature of this complex of file structures is that the student or the researcher working at a teletype may refer to studies, variables, categories within variables, etc., by meaningful labels; from these labels, the system can find in the header records the corresponding encoded location of the items requested. Thus, one avoids the customary irritation imposed upon the researcher and the total confusion imposed upon the student; no one ever enjoyed thinking about the 3-punch in column 62 of card 5, and we are happy to be done with this approach.

As indicated above, all data are stored in the rawest possible state so that the user may tap whatever richness of data exists. However, it is rare that one has any use for the raw data. For example, consider the 3-digit code which the Census Bureau provides for occupation. . .

000	Accountants and auditors
010	Actors and actresses
012	Airplane pilots and navigators
⋮	
⋮	
985	Laborers (not elsewhere classified)
995	Occupation not reported

The code numbers have no intrinsic value, and there are far too many of them to use in a tabulation. Therefore, header records are maintained in which are stored those numeric parameters and those string-variable labels necessary in order to reduce occupation to either a "standard group" or to a "standard dichotomy." For example, the following information permits translation of the raw 3-digit code into a "standard group". . .

Punches	"Standard Grouping" for Occupation
000–199	Professional, technical, and kindred workers
200–249	Farmers and farm managers
250–299	Managers, officials and proprietors (excluding farm)
300–379	Clerical and kindred workers
380–399	Sales workers
400–599	Craftsmen, foreman, and kindred workers
600–799	Operatives and kindred workers
800–809	Private household workers
810–899	Service workers (excluding private household)
900–959	Farm laborers and foremen
960–994	Laborers (excluding farm and mine)
995–999	Occupation not reported

The obvious solution to this 3-digit mess—using the hundreds digit to identify groups—is inoperable; for example, there are 200 possibilities for "Operatives" and only ten for "Private household workers." Thus, the header records are used to reduce occupation to a "standard group." In like manner, the header records contain the information necessary to reduce occupation to the "standard dichotomy" of blue collar-white collar.*

Data retrieval and reduction

Having a variety of social science data files available on-line in a time-sharing system is not a particular advantage unless one can retrieve and manipulate these data with ease. First, the point has been made that—within a data file—the user can work with only a subset of available variables at any one time. The selection process is more complex than this, for there should also be the capability of selecting a subset of observations.

One class of selection procedures is statistical sampling. This is particularly useful for pedagogical purposes and includes (a) systematic sampling, (b) simple random sampling, with or without replacement, and (c) stratified sampling. Another group of selection methods involves the definition of subsets; for example, with reference to the Census file, one might be interested in examining data for all adult Negroes living in the south. Mechanisms for logical inclusion and/or exclusion—at either point in the two-stage system—are imperative for social scientists. A third selection device is needed to retrieve a specific datum; when observations can be identified (which is not the case with Census data, since respondent anonymity is well-protected), the datum for a specific variable for a specific respondent is retrieved.

To summarize, there are a variety of selection devices. The user selects one data file from those available, and he selects a subset of the variables contained within that file. Furthermore, he may institute a sampling procedure, and he may define logical subsets via inclusion and/or exclusion. Finally, he may select a specific datum for retrieval.

In the retrieval and reduction phase, as was suggested above, the user may utilize either the "standard group" or the "standard dichotomy" built into the header record. These are devices for grouping raw data into more meaningful, manageable categories. However, the groupings provided by the system may not be suitable to the needs of a particular individual. With respect to categorical data, there are options for re-mapping or re-grouping of variables according to the user's needs. In addition, he may combine several existing variables and define a new computed variable. For example, income, occupation, and education might be combined to form a measure of socioeconomic status.* With respect to continuous variables, certain transformations (addition/subtraction of a constant, multiplication/division by a constant, log transformation, etc.) are available as standard functions. Although the array of data-manipulation devices currently available may not be exhaustive (even if this is our goal), that array clearly is suitable to the large majority of demands made of the system.

Tangential to the question of devices for data manipulation, the user may maintain a private file of re-mapped variables, new computed variables, and transformed variables. At that phase of execution of the system when he is asked to identify variables and their mapping mode, the user can direct the system to his private file to obtain the data-manipulation devices defined there. This technique not only eases the user's task; it also both reduces the chance of error and provides consistency from one run to the next.

Thus far, the task of data reduction has been viewed as the construction of a large cross-classification table. Actually, there are three devices for data reduction: (1) cross-classification tables, (2) sums, sums of squares,

* Actually, some "standard dichotomies"—such as the example here, occupation—are treated as trichotomies. Since roughly half of the population is not in the labor force (housewives, children, retired persons, etc.) it is necessary to have a third category for this "dichotomy": not in labor force.

* In the 1960 Census data, there already is a combined measure of socioeconomic status provided by the Census Bureau. In this instance, the ability to generate a computed variable would be useful if the individual wanted a measure of socioeconomic status different than the one provided.

and sums of cross-products, and (3) the combination of both. Each of these is discussed in some detail, as follows:

(1) Cross-classification tables provide a device for reducing large quantities of raw discrete data to a file small enough to be manageable in time-sharing mode. Specifically, we want to accomplish a reduction to 4,000 cells or less. Let it be clear that no one would really want to see the table with 4,000 cells; of interest are the numerous possible sub-tables which can be quickly generated from this massive one. It is a curious phenomenon, therefore, that some feel seriously restricted by such a limit. However, when the sample size is less than 4,000 observations, instead of entering cell frequencies into the working file, it is more efficient (in terms of information transmitted) to enter the *cell number* of each observation.** The generation of a table apparently so complex sounds as if it is not worth the effort, yet it is important to remember what is being accomplished by such a procedure; more information is being tapped per (time-consuming) access of the storage device. Since the working file can be saved, it could be days or weeks until such a large quantity of information is exhausted of content. The limitation to this approach is that it is a viable one only if the sample size is less than 4,000.

(2) Generation of sums and sums of squares and cross-products is a second device to reduce masses of raw data. From such a reduced file, one can perform correlation analysis, regression analysis, factor analysis, etc.† For instance, the simultaneous examination of fifty or sixty variables is not an unreasonable task. The difficulties encountered in this area are concerned with issues other than magnitude.

It is often the case that our data are incomplete; if someone chooses not to answer an item on a questionnaire, there is no way to extract the data from him. In practical terms, what can be done when that respondent's data are processed: One possibility is to discard the entire observation whenever one or more items of data are missing; customarily, this means a loss of at least fifty percent of the sample. Another possibility is to insert the mean from observations without missing data; although this approach takes advantage of all existing data and minimizes bias introduced, it creates havoc with respect to customary statistical techniques. A third approach is to replace missing values with random variates generated from a distribution comparable to the one observed with those data which are not missing; in subsequent multivariate procedures, one finds the covariance unnecessarily decreased. A fourth approach is to carry out repeated large regression analyses on existing data in order to predict values for missing data; here, one is bound to encounter almost all of the pitfalls and disadvantages of regression analysis. A fifth possible approach is being explored and involves the generation of sums and sums of squares and cross-products everywhere that data are available. Of course, a vector (for the sums) and a matrix (for the sums of squares and cross-products) of subsample sizes must be generated. There are serious statistical problems with this approach, but we are convinced that this is the most fruitful avenue to explore.

The second major problem with continuous data can be a serious one. If the individual is interested in examination of residuals in multiple linear regression analysis, then a second accessing of the raw data is required. It is mathematically impossible to compare predicted and observed values (at the level of the observation) from summary data; so examination of residuals forces a second reading of raw data.

(3) The combination of categorical data (cross-classification tables) and continuous data (sums and sums of squares and cross-products) represents analysis of variance problems and multivariate attribute analysis with dependent means. In combining the approaches of the first two reduction techniques, we are actually combining their limitations and problems. Work will not start in this area until the first two reduction devices are operating smoothly.

Now that we have covered—even if with brevity—the major points of data storage, retrieval, and reduction, let us summarize some important ideas in order to make an additional point. Data are stored in two formats, bucket and vat. Retrieval-reduction may be accomplished in three major ways, and there are several alternatives within the main ones. It is the responsibility of the system and *not* the user to determine which combination of alternatives is the most efficient to utilize. Since the working file can be in one of several forms, the first item in the file is a key to the form utilized. Only the highly sophisticated user will ever

** That is, suppose one wanted to construct a cross-classification table from 17 trichotomous variables. This represents 129,140,-163 cells. If the sample size is less than 4,000 then one might generate a working file consisting of less than 4,000 such that each entry represented a cell number (ranging from zero through 129,140,162). Note that "bucket" format would be necessary for a task of this nature.

† It can be argued that analysis of variance, regression analysis, analysis of covariance, and other procedures are special cases of the general linear model [James Fennessey, "The General Linear Model: A New Perspective on Some Familiar Topics," *American Journal of Sociology,* 74 (July 1968), 1-27]. While this notion cannot be challenged in theory, it is clear to us that—in terms of the mundane task of data retrieval, reduction, and analysis—such an approach is not practical.

know that these alternatives exist; the more typical user will never know about these considerations.

Data analysis

There are three major approaches to the question of analysis of data, once retrieved and reduced: (a) a standard teaching package, (b) a standard research package, and (c) an open-ended approach.

The teaching package consists of a library of conversational-mode programs designed to lead the student through the process of data analysis. These are not Computer-Assisted-Instructional programs; rather, they are simply conversational-mode routines. The student is asked—for instance—whether he wants tables percentaged horizontally, vertically, or not at all; whether he wants to examine expected cell values, the matrix of differences between observed and expected cell values, or neither; what level of measurement (nominal, ordinal, etc.) his data represent and whether he wants a measure of association appropriate to that level, etc. We have attempted to design these programs so that the student has the opportunity to make bad decisions; wherever possible, we remind him of what he is doing with brief, diplomatically-phrased statements. Also, we consciously limit the beginning student to a very few, widely-accepted measures of association so that he will become familiar with data analysis and the process of posing substantive questions to empirical data; there is no point in terrorizing him with the multiplicity of possible measures and techniques which are less than widely accepted and used. Furthermore, as he attains a certain level of sophistication and grows weary of the time-consuming conversational-mode programs, he is free to turn to the research package.

The research package is a series of programs like the teaching package, but the routines provide more of the diversity of available methodologies and also avoid the conversational mode (in the sense in which it is used in the pedagogical programs). The intent of the research package is to provide the most information in the least amount of teletype time. The problem encountered here is to find the right balance of information transmitted and time taken to do so. That is, one could devise a routine for the analysis of bivariate contingency tabulations which would provide every known measure of association and every possible significance test, yet the teletype time required to print out all of these values would be such that no one would want to use the program. Thus, the research package is designed to keep most of the users happy most of the time.

Finally, there is an open-ended approach available to sophisticated users so that they may devise their own analysis programs. In essence, such persons are taking the working file and doing their own programming. The individual who wants to use some technique or some measure which is too bizarre to be included in the research package still has the opportunity to get his work done within our system. Also, the researcher working on new techniques and new measures can write his own programs and then apply them to the diversity of data files available on line. Providing this open-ended analysis option is trivial for the systems designers, and its availability provides true flexibility to those using the system.

CONCLUDING REMARKS

We have attempted to build a data storage, retrieval, reduction, and analysis system which provides the benefits of time-sharing computation to social scientists. The researcher can perform sophisticated, complex manipulations of his data with a minimum of time, energy, and system familiarity. Through the very same vehicle, the beginning student can—in a somewhat limited but guided manner—experience the activities of the researcher in working with empirical data. There are several important implications of such a system.

First, with even a modest amount of initiative on the part of our students, we will experience significant challenges in our lecturing. That is, when a student reads that thus-and-so is true or hears in a lecture that something is true and is not personally convinced, he will have the capability to test with empirical data the statement in question by investing fifteen minutes or less of his time at a teletype. It is conceivable that the availability of the system described in this paper will contribute to an intellectual dialogue between faculty and student.

Second, because time-sharing reduces turn-around time to zero (by definition), the researcher will be able to interact with his own data. In a batch processing system, by the time the necessary control cards are punched and the routines executed and the output examined, it is almost possible to forget some of the details of the hypotheses being tested. In a time-sharing social science package, the results of one run might easily suggest or dictate the substance of the subsequent run. As the researcher or the student has the substantive question in mind, he can explore what happens to initial relationships as various controls are introduced.

Third, when the user is doing original work with a particular data file and encounters an unusual or unique relationship among certain variables, he will be able to attempt to replicate his findings by generating comparable runs on other data sets. As has been repeatedly noted in recent years, one of the important advantages of having data libraries in the social sciences is the ability to try to replicate tentative results without the tremendous expense of returning to the field to acquire more data.*

* This is a matter of serious concern to the social scientist, since he does not have the laboratory and the small army of laboratory assistants to attempt replication for him (usually in a relatively short period of time and at a modest cost). To repeat a large survey research effort would require tens or hundreds of thousands of dollars and months or years of time.

Finally, it is the case that work which would have required days or weeks of time with more traditional techniques (such as the IBM 101 Electronic Statistical Machine), even if carried out by a skilled Ph.D., can now be accomplished in roughly thirty minutes by an intermediate-level undergraduate working at a teletype. What was once a research project is now a small assignment to be carried out by undergraduates.

Secondary school use of the time-shared computer at Dartmouth College

by JEAN H. DANVER and JOHN M. NEVISON

Dartmouth College
Hanover, New Hampshire

INTRODUCTION

Soon after the first Dartmouth Time-Sharing System began operation in May of 1964, the local high school installed a teletype. Within two and one-half years, eight high schools had tied into the Dartmouth System. The effects were startling. Hundreds of students were taught the BASIC language. Some of them produced highly sophisticated programs. Teachers were using computer applications in mathematics and science courses. These effects convinced Dartmouth of the high value of computing in secondary education. However, the results were not sufficiently examined. There was no documentation on recommended procedures nor written units of classroom applications. Therefore, in the spring of 1967, Dartmouth College proposed that the National Science Foundation support work in developing, expanding, sharing, documenting, and publishing the results of computing experience in secondary schools.

This proposal was accepted and funded in part by the National Science Foundation.* Eighteen secondary schools are now involved with the College in this two and one-quarter year project begun in June, 1967.

The main purpose of the project was stated in the project proposal:

> "... to demonstrate the large-scale use of the computer as a broad aid to secondary education without requiring major curriculum changes or extensive teacher retraining. Through materials to be developed cooperatively with the participating schools, we expect to show the value of computing as an aid to course teaching in many subjects, and as a significant mechanism for extracurricular education of students....
>
> These materials ... will provide important guidelines for the development of the potential of computers in secondary education on a broad front."

The project was designed to meet the following objectives:

- To demonstrate that computing can be useful in teaching other high school subjects as well as mathematics.
- To demonstrate that computing can encourage students to think creatively.
- To experiment with classroom techniques and to suggest practical methods for integrating computing into course work.
- To publish materials which will serve as guidelines for other undertaking projects to develop the potential of computers in education.

These objectives are reviewed in Appendix B.

In accomplishing the goals of the project, Dartmouth became a regional computer center for a group of secondary schools. How Dartmouth did this, the costs involved, and the effects in the secondary schools themselves are the main topics of this report. First to be considered is the regional system.

The system

The schools

The project schools are listed in Table I along with their location, ninth through twelfth grade school population, size of their 1968 graduating class, and the percentage of the graduates who went to a four-year college.

There are 12 public schools and six private schools spread over a six-state region. This group of schools represents a varied sample of American high schools. They range from small rural schools to large city

* Funded under the terms of Grant NSF GW-2246.

Table I—High schools participating in Dartmouth's NSF secondary school project

School	*Location*	*Population*	*Graduating Class*	*Percent to 4-year College*
* Benjamin Franklin High School	New York, New York	3,200	190	47.4
* Cape Elizabeth High School	Cape Elizabeth, Maine	573	116	48.3
* Concord High School	Concord, Massachusetts	1,478	333	37.5
* Hartford High School	White River Junction, Vermont	516	135	24.4
* Hanover High School	Hanover, New Hampshire	679	110	61.8
* Keene High School	Keene, New Hampshire	1,415	284	33.1
* Lebanon High School	Lebanon, New Hampshire	536	152	34.8
+ Loomis School	Windsor, Connecticut	444	100	100.0
* Manchester Central High School	Manchester, New Hampshire	1,616	356	41.6
* Mascoma Valley Regional High School	West Canaan, New Hampshire	456	61	29.5
+ Mount Hermon School	Mount Hermon, Massachusetts	620	177	99.4
+ Phillips Academy	Andover, Massachusetts	860	242	94.6
+ Phillips Exeter Academy	Exeter, New Hampshire	789	252	97.6
* Rutland High School	Rutland, Vermont	1,063	196	32.6
# St. Johnsbury High School	St. Johnsbury, Vermont	701	149	32.2
+ St. Paul's School	Concord, New Hampshire	458	95	100.0
* South Portland High School	South Portland, Maine	1,700	335	50.1
+ Vermont Academy	Saxtons River, Vermont	217	65	95.4

* Public
+ Private
# St. Johnsbury is considered a public school for the purposes of the project. It is the sole high school for the town of St. Johnsbury, Vermont. It operates its teletype from 8 a.m. to 4 p.m., Monday through Friday, as do all the public schools. Private school hours are 8 a.m. to 8 p.m., Monday through Saturday.

schools to highly specialized private schools. Six of the public schools are large city schools with populations of over 1,000 students. The rest are smaller rural and suburban community schools, one of which is in a college community (Hanover High School). The public schools are less oriented towards higher education than the average American school. On the average, just under 40 percent of the public-school students go on to four-year colleges compared to the 1965 national average of 53 percent. Ninety-eight percent of the private-school graduates attend four-year colleges.

Hardware

The schools are connected to the Dartmouth/GE-635 Time-Sharing System via long-distance phone lines. The majority are utilizing the Model 35 KSR teletype of the Bell Telephone System. The schools all had one teletype installed for the year 1967-68 with two exceptions: Mt. Hermon School and South Portland High School had an additional teletype installed which they completely supported with their own funds. The 1968-69 school year should see additional teletypes placed in three more schools.

There were no unusual hardware difficulties experienced by any of the schools. Most operational problems were related to the central system at Dartmouth. Consequently, they affected all users.

Teacher training

The quality and the quantity of the machine usage contributes a large amount to the success of any computer project. It was felt necessary to have at least one teacher in each school who was proficient enough with the BASIC language and the hardware to feel at ease teaching it to students and fellow teachers. With this end in mind, during June of 1967, each school sent one teacher to a four-week summer training program at Kiewit Computation Center. These initial sessions were taught by Mr. John Warren of Phillips Exeter Academy. He was an experienced two-year user of the Dartmouth computer in a secondary school environment. The teachers were instructed in the BASIC language and possible classroom applications. They also received suggestions on handling teletype breakdowns and telephone problems, and on method of administering student use of the teletypes.

Our first year documentation shows that these sessions were very successful. However, it was found that during the school year, few additional teachers learned to program the computer. The main reason for this was that almost all available "hands-on" time was taken by students. Teachers need much more time at the teletype than students to learn the BASIC language. Consequently, Dartmouth decided to hold two two-week teacher training programs during the summer of 1968. The purpose of these training sessions was to raise the number of teacher users to at least four in each school. Four enthusiastic teachers can be much more influential with teachers and administrators than one. More teacher users should greatly increase the classroom use of the computer as well as the number of students trained in the language. (Early reports from teachers indicate that this is exactly what is happening.) A core of teachers should be a big factor in perpetuating good computer use in the schools. No matter how successful the present students are, they are only transients and can have little effect in the continuation of their efforts. Teachers, a more permanent part of the institution, can.

Our experiences training teachers have taught us several things:

- Mathematics teachers learn to program a computer easier than teachers from any other high school subject area.
- Science teachers, especially those who teach chemistry and physics, learn fairly easily, also.
- Social studies teachers have the most trouble learning.
- Younger teachers learn easier than older teachers.
- Experienced teachers are more successful back in school than beginning teachers.
- There are exceptions to all of the above.

These observations are what one would normally expect. Mathematics and science teachers are familiar with the kinds of thinking necessary to successfully program a computer. Social studies, as it is taught in high schools, does not demand this kind of thinking. Those who normally teach computer training are mathematically-oriented themselves and, hence, are not sensitive to the problems encountered by a humanities teacher. Younger people are often more adaptable than older ones. They are less experienced and hence more willing to try new things. We recommend to any school who is thinking of obtaining access to a computer that they will improve their chances of success by training a young, successful mathematics teacher, who does not already have extra duties such as department head. Then, give him a reduced teaching load so he may adequately perform the extra duties involved.

Special services

As a regional computer center for secondary schools, the Kiewit Center provides a number of special services for the teachers and students. Some of these activities are designed to help give better computer service. Others were initiated to promote project ideas and still others are provided just for fun. Special services can be grouped in the two categories of problem solving and communications:

1. Problem Solving—The Kiewit staff stands ready to assist the schools in any way possible. All telephone and teletype malfunctions are reported to the Computation Center. Not only can we better diagnose problems (which often turn out to be at the Center), but we receive faster service from the phone company besides. Center personnel are also on hand to solve any questions concerning programming or to provide the use of peripheral equipment such as the high-speed printer, card reader, or card punch. The project Coordinator handles any problems associated with the project itself.
2. Communications—Providing communication between the Center and the project members and among the members themselves proved to be important not only to the success of the project but to the success of secondary use as a whole. The spreading of ideas serves to test their value through repeated trials and to provide motivation for creating new ones. Teachers' time is very valuable. They do not have secretaries nor mailing budgets. They cannot be expected to keep up contacts on their own. Several methods were used by Kiewit for providing communication among the Center, the schools, the teachers and the students with varying degrees of success.

These were:

- Teachers' gossip file—This was a file in the system which the teachers could call up to list or to add a message. It was not a very successful communicating device among the teachers. They had difficulty competing with the students for terminal time. When they did find themselves at a terminal, they did not want to spend the time required to list the file.

- Student gossip file—This is a similar file for students, but much more successful. They exchanged school news and swapped ideas on programming problems.
- Biweekly Bulletin—This is a bulletin published every two weeks and sent out to all the schools. It includes original students' program descriptions, descriptions of teachers' class usage, and other items of interest to secondary school users. Most student work published in the Biweekly is submitted as entries for the Kiewit Cup Contest. This is a contest sponsored by the Center which presents citations to individual students who submit outstanding programs and awards a cup to the school which "demonstrates the most outstanding use of the computer." This contest in 1967-68 served a purpose for the project by providing a sample of the work the students were doing on their own. We have maintained it through the second year because of widespread student interest.
- Teachers' conferences—Twice each year the Center holds a Saturday conference for participating teachers. These enable all the teachers to get together and discuss problems of common interest and hear first-hand about new developments at the Center. Common procedures include open discussions, presentation of classroom work, and guest lectures. The conferences are very popular with the teachers.
- Student conference—This was a one-day conference organized by a group of students and the project Coordinator. The students came from all the project schools with a teacher. Most of the speakers were fellow high school students.
- Teacher newsletters—Letters were sent containing information on conferences, school visits, and other subjects of interest only to project teachers.
- Visits to the schools by the Coordinator—These proved to be necessary during the first year to ensure that all was progressing smoothly. Some teachers chose not to communicate frequently with the Center. They would make suggestions and ask questions in person, while reluctant to phone or write. The visits also gave the Coordinator a chance to see personally what was going on in every school and to meet and talk with students.

Costs

Table II gives a cost analysis for the schools. It shows that the overall cost of a one-teletype operation in the 1967-68 school year ranged from approximately $4,550 to $12,260 a year.* It should be noted that actual computer time is about 60 to 65 percent of the total cost. The remaining costs are telephone charges.

There is a definite difference in costs to the private and public schools, the private costs being higher. The main reason for this is that private schools' teletypes are available for student use about 32 hours more per week than in the public schools.

In C, all the examples are of public schools except for the typical private school and the highest cost school. It should be noted that among the private schools, the range in computer costs is only $573.36 while the range in overall costs is $2,230.87. All these schools are using their terminals at close to the same number of hours and effectiveness. The cost difference comes with communications, which is a function of the distance from the Kiewit Center. At the present time, differences in school population are not reflected in cost or usage. With just one teletype, it does not take very many high school kids to keep it going all of the available time.

It should be noted that the trend at Dartmouth, as in other time-sharing installations, is towards lower computer rates. In the fall of 1968, Dartmouth's rates for computer time were reduced approximately 40 percent for the secondary schools. This should reduce the overall charges by a factor of 25 percent for the 1968-69 school year. Also, under development are various methods for combining several teletypes on one telephone line. This seems to indicate that communications costs will also lower.

The project's overall computer costs are somewhat misleading. Some of the schools are a long distance away. Under normal circumstances they would be tied into a closer installation. Also, the current rates reflect the costs of some of the special services Dartmouth provides and the small number of commercial users on the system.

According to 1965 salary figures for the states involved, a Dartmouth teletype connection last year cost $200 to $1,500 a year more than the average teacher. Is it worth it? To answer that question, we should look at what is happening in the schools themselves.

The schools

To answer the question "Is it worth it?" we have to look at the average high school student sitting at a tele-

* It should be noted that the actual cost to the schools ranged from $0 to $5700. Sixty-five percentof the total computer and communications costs were financed by the NSF and Dartmouth College.

Table II—Cost figures for the secondary schools

	Public	*Private*	*Combined*
A. Average Cost (9-month year)*			
Overall cost	$6,226.69	$8,912.63	$ 7,074.88
Communication (tty)	$2,536.34	$3,119.21	$ 2,720.40
Computer	$3,690.35	$5,793.42	$ 4,354.48
Overall cost/hour	$ 4.94	$ 5.34	$ 5.07
Actual cost/school/terminal	$1,563.39	$4,280.09	$ 2,469.47
Actual cost/hour/terminal	$ 1.24	$ 2.57	$ 1.77
B. Average Usage			
Number of users/month	68.90	127.00	88.30
Terminal hours/month	139.92	185.36	155.07
Terminal hours/user/month	2.03	1.46	1.75

C. Examples of Overall Yearly Cost Per Terminal*

School	*Communications*	*Computer*	*Overall*
Lowest	$1,366.00	$3,275.40	$ 4,541.40
Highest	$3,335.32	$8,921.18	$12,256.50
Typical Private	$3,901.65	$6,169.23	$10,070.90
Small Rural Local	$1,417.86	$3,402.22	$ 4,820.08
City	$2,923.51	$3,848.76	$ 6,772.27
Suburban	$2,509.38	$3,603.57	$ 6,112.95

* 1967–68 figures. 1968–69 figures will be about 30 percent lower.

type. When the student acts creatively and intelligently then the computer can be a *creative extension of his intellect*. Why is this so? If something is to be produced, he must teach the machine to produce it. The value of Dartmouth Time-Sharing to the student is in direct proportion to the value of the tasks the student sets for the machine. The computer, when used well, can serve as a catalyst in releasing his creativity.

The use of the Dartmouth computer in the project schools as a rule in no way relates to what has commonly been termed CAI (Computer-Aided Instruction). In our case the student takes complete control of the machine. The success of this project hinges on what the student does with the computer, *not* what the computer does with the student.

Getting the students started

Each school introduced BASIC to the students in their own way. Several good write-ups on these different methods are included in the Topic Outlines.*

* See Appendix A.

While teaching students to use the computer, some interesting observations were made:

1. Students need only about 20 minutes per lecture or about two hours total terminal time to learn to use the machine (compared to 20 hours for teachers).
2. Whole courses devoted to programming are obsolete. Students are more successful working with their own problems. They learn BASIC so quickly that any necessary instruction time is easily included as a topic in a standard high school course. There are courses taught in some of the project schools which are computer oriented. However, these courses do not stress computer techniques. Rather, they present math best taught with the aid of a computer.
3. Time spent teaching BASIC in many regular courses was not time lost. Teachers found that they actually covered more material by the end of a term. Less time was spent on tedious calculations. Less time was needed for drill work because programming a concept demands complete understanding. Complex problems pre-

viously "accepted" could be demonstrated with success and understanding.

4. There was no grade level found *best* suited for teaching students how to use the computer. One of our teachers taught elementary BASIC programming to a group of talented fourth graders. Several schools taught BASIC to seventh graders and found that the *average* seventh grade student can learn to program.
5. No ability level has been found below which students could not learn to program. This is not to say all students learn with equal ease or are equally interested. However, computer work has shown itself not to be one of those activities which only the brightest of the college bound can handle.

Student use

Students used the computer both inside and outside of their classrooms. They used it for assigned computer work, for doing homework, for projects, and for fun. Most of the student programs entered in the Kiewit Cup Contest were written just for fun. These programs, representing a small fraction of the students' actual work, are quite commendable. A short summary of the list of Kiewit entries could hardly do justice to the imagination and ability of these students who had limited access to a computer over a short nine-month period. These entries are substantial evidence that, given an opportunity to follow his own interests and inclinations, a student will use the computer as a *creative extension of his intellect.*

The list of Kiewit Cup programs includes programs in the mathematical areas of number theory, algebra, plane and analytic geometry, calculus and probability and statistics, and in the science areas of chemistry and physics. Also, a small scattering of programs in almost all other academic subjects touched upon in high school is included. In addition, there is a large group of game playing and other miscellaneous programs.

The number theory and algebra programs were the most numerous. These included all the usual topics such as factoring, prime numbers, graphing, equation solving, and the like—many of which were very sophisticated. One ninth grader used Newton's method to find the square root of a number and developed a similar method of his own to find cube roots. Calculus programs included calculating limits, areas under a curve, and differentiation. One of the area programs utilized Simpson's Rule and another used random proportioning and Rieman Sums. One student submitted a clever approximation of π by Monte Carlo methods. The geometry, and probability and statistics submittals covered the standard calculations found in those fields. There were several very nice probability simulations. Nice simulation programs were also found among the physics and chemistry group. One student wrote a program to predict the orbit of the "Syncom" satellite. Another program teaches the valences of chemical radicals. A couple of the 27 different game submittals were considered so much fun that they were put in the Kiewit game library.

More than any other group, perhaps, the miscellaneous programs are the best indication of what high school students can do. Some of these programs write poetry, compose music, teach other students topics in foreign languages, or science or grammar. They score games, analyze elections and surveys, layout the school newspaper, and compute payrolls. The topics are just too numerous to list. Some high school students became very interested in systems problems. One wrote his own abbreviated BASIC system and another developed a TRACE system for locating his programming errors. An eighth grade student wrote an excellent interpreter of LISP in the BASIC language. Anyone using LISP in the Dartmouth System uses his interpreter.

The winners in the Kiewit Cup Contest included students from seventh through twelfth grades and from all supposed ability levels. Some of them have gone on to college, while others were even sent to reform school. Some were repeating a course when a use of the machine caught their interest. Some have never shown much interest in anything associated with school before.

Students seldom allow the teletypes to sit idle at their high schools. The statistics on usage (Table II.B.) hardly leave enough idle time for standard hardware failures and occasional signing on and off between users. These student users are not just a small hard-core, either. We have found that the number of students using the machine, at least occasionally, is around 25 percent of the total high school population. This figure is extraordinary when you consider that, for most of the schools, there was only one person to teach them the BASIC language.

Classroom usage

One of the project's objectives is to demonstrate that the computer can be a significant contribution to class work already found in existing school curricula. Classroom use was significant. Our first-year teachers wrote a set of Topic Outlines documenting their class use. Each outline is an explanation of how they used the computer in conjunction with regular classroom work. These outlines range from a three-class demonstration to a whole-semester course. Their titles are listed in Appendix A. Since the majority of the first-year teachers were math teachers, most of the topic outlines concern mathematics. There are several explaining science usage and three or four in other fields (notably business and teaching BASIC).

Because of the fourfold increase in project teachers, this year should see a large increase in class usage and documentation, especially in fields other than mathematics.

During the 1967-68 year, the following class-usage patterns have emerged:

* Mathematics use is heavy. However, some courses see more usage than others (calculus, for instance, as opposed to geometry). The computer lends itself to convincing classroom demonstrations in many topics such as logarithms, limits, and equation solving. Teachers use it to illustrate concepts usually taken on faith because of the massive calculations involved. Some courses were expanded to include topics which were easily adaptable to computer applications (matrix algebra for example).

Teachers note an immediate transfer of enthusiasm from a successfully written computer program to the mathematics involved. Many more students than ever before are going beyond their normal classroom work and studying advanced concepts on their own. Concepts which they needed to know in order to write a program. Many teachers are turning to program writing as a method of teaching in mathematics courses which stress calculations. (Algebra, trigonometry, statistics, and calculus, for examples.) Writing and successfully running a program increases the students' understanding of a mathematical algorithm and saves class time.

Science usage is quite substantial and is increasing. Much of the use to date is in the chemistry or physics lab. The computer performs tedious calculations and computes class statistics. Students see that averaging all their results together produce better approximations to the predicted value. Teachers have found that the Chem Study and Introductory Physical Science labs lend themselves easily to computer work.

A superb use of the computer in science classes is in answering those spontaneous what-if questions that would normally be impossible to answer because of the calculations or lack of time. A computer demonstration can be modified in a matter of seconds or students can write a program themselves to answer the question.

. . . A senior astronomy student asked how often the nine planets would line up. The class wrote a program (assuming all the planets were on the same plane) and found that it would happen once about every 65 million years.

. . . One ninth grade class became concerned about comets. As a class they decided to write a program to determine the orbit of a comet. The result was significant learning about comets, programming, and about forces between moving bodies.

The computer has been used some in the simulation of science experiments. Notable in this area are population and genetics experiments in biology.

The computer class uses in other subject areas have been more sporadic. Some language teachers use some CAI-type programs for drill. Some social studies classes have done election surveys and analyzed the results via computer. One English teacher uses it to help students learn literature concepts and write creatively by generating random metaphors and phrases. However, most of the work in the social sciences and humanities was done by students for special projects or on their own.

SUMMARY

Eighteen secondary schools throughout New England are involved with Dartmouth College's Kiewit Computation Center in a two and one-quarter year project supported by the National Science Foundation. The main purpose of the project is to demonstrate that the computer can be a significant contribution to secondary education within the existing curriculum and without extensive teacher retraining. Important outcomes are the demonstration that computing encourages students to think creatively, and the publishing of materials to serve as guide lines to others in the utilization of computers in education.

In its role as a regional computer system, Dartmouth provides may auxiliary services for the schools. These include aid with malfunctions and programming problems, fostering communication among the participants, and sponsoring conferences and training sessions.

After its first year, the project has been highly successful. Hundreds of students have been trained in BASIC and are using the computer creatively on their own. Teachers have written a considerable number of outlines for classroom work and 1968-1969 will see the results of the previous year continued with greater emphasis on materials for science, business, and below-average students. The project findings are being published for general distribution to those who are interested in the potential of computers in education.

APPENDIX A

Index of topic outlines according to grades

Grade	*Title*
4	Four Classes with Fourth Graders—Introduction to BASIC William A. Smith, Lebanon High School
7–12	Some Suggestions for Student Programs Jean Danver, Dartmouth College
7–12	A BASIC Manual for High School Students (with exercises) Floyd McPhetres, Hartford High School
7–9	Junior High School Uses of a Time-Shared Computer G. Ralph Bolduc, Cape Elizabeth High School
7–12	Some Computer Applications in Secondary School Science Spencer Laramie, Mascoma Valley Regional High School
9	Two Examples of Linear Programming in an Algebra I Class William Smith, Lebanon High School David Penner, Phillips Andover Academy
9	Solution of Simultaneous Linear Equations John Conover, St. Johnsbury Academy
9	BASIC in 10 Minutes a Day Louis Hoitsma, Phillips Andover Academy
9	Introduction of the BASIC Language, Teletype Usage, and Elementary Programming Peyton Pitney, Mount Hermon School
9	Ninth Grade Word Problems Warren Hulzer, St. Paul's School
9–10	Random Sample Studies Charles A. Tousley, Keene High School
10	The Binomial Theorem Gary Toothaker, Vermont Academy
10	Genetics of the Fruitfly—Phenotype Ratios Charles A. Tousley, Keene High School
10	The General Solutions of the Quadratic Equations G. Ralph Bolduc, Cape Elizabeth High School
10–11	Value of Cos (t_1) (An Iterative Technique) John C. Warren, Phillips Exeter Academy
10–11	The Circular Function John C. Warren, Phillips Exeter Academy
10–12	The Use oft he Computer in Air Pollution Study John Conover, St. Johnsbury Academy
10–11	Summer School Computer Course John Hauber, Loomis School
10–11	Slope of a Line and Common Solutions for Systems of Linear Equations G. Ralph Bolduc, Cape Elizabeth High School
11	Areas and Perimeters of Circles and Ellipses Paul Kenison, Manchester Central High School
11	Slopes of Exponential Functions George H. Lewis, Concord High School
11	Five Ionization Reaction Problems Spencer Laramie, Mascoma Valley Regional High School
11	Area Under Trapezoid Charles A. Tousley, Keene High School
11	Introduction to Logarithms Charles A. Tousley, Keene High School
11	Finding Nth Degree Equations from a Set of Tabular Values Paul E. Kenison, Manchester Central High School
11–12	Finding Approximations for Irrational Zeros of Polynomial Functions Peyton Pitney, Mount Hermon School
11–12	Three Simple Examples of Computer Use in a Physics Laboratory John Martin, Rutland High School
12	Using a Time-Shared Computer in Developing the Law of Sines, the Law of Cosines, and the "Solution of Triangles" Floyd McPhetres, Hartford High School
12	Free Falling Bodies and Projectile Motion Spencer Laramie, Mascoma Valley Regional High School
12	Computer Course for Business Students Ann Waterhouse, South Portland High School
—	An Adult Education Course in BASIC Programming John Martin, Rutland High School
8–12	Collected Uses of a Computer in Probability and Statistics Mary Hutchins, Hanover High School
12	Two Programs on Riemann Sums George R. Smith, St. Paul's School
12	Numerical Integration G. Albert Higgins, Jr., Mount Hermon School
11–12	A Unit in Matrix Algebra Ann Waterhouse, South Portland High School

APPENDIX B

Summary of NSF-proposal objectives

I. *Meeting Objectives*

Following is a summary of the project finds to date, as they relate to its objectives as set forth in Part II of the initial project proposal.

1. *To demonstrate that computing can be useful in the teaching of subjects other than mathematics.*

   The computer has been of use in many subjects other than mathematics as testified in the main body of this paper. These subjects include all of the high school sciences, modern languages, social studies, English, and several business courses.

   One significant obstruction to demonstrating the computer's usefulness in teaching in the social sciences is the average social science teacher's inexperience with statistics. Indeed, some high school students have a much better grasp of applied statistics and their use on the computer, than many college professors of the social sciences.

2. *To demonstrate that computing can encourage the student to think creatively.*

   Literally hundreds of programs received from students testify to the student's creative ability. The very nature of going from a rough idea to an articulate set of specific directions in a working program is in itself a highly creative act.

   Several of the more complicated games written by some advanced students as well as some of the less involved programs written by slow students may stand, each in its own way, as the most outstanding intellectual creation that student will make in his four years in school. This claim is backed by the fierce pride students have in their programs. They are their very own. They made them and they work.

3. *To demonstrate that computing can be effectively introduced into secondary schools without extensive curriculum changes or teacher training.*

   The 1967 four-week training session for teachers was shortened to two weeks in the summer of 1968. This speaks for itself. Commentary on teacher and student training during Teachers' Conferences also confirm the ease and speed with which computing can be introduced into schools.

4. *To experiment with techniques for introducing computing to the student, and for helping the teacher integrate computing in his courses.*

   Techniques for introducing the computer abound in the Topic Outlines and are discussed in the body of the First Year Report. Over 30 Topic Outlines written by teachers themselves we now have available.

5. *To develop materials that will aid other schools to take full advantage of the opportunities computing provides.*

   The report on a Four-Week Training Session for Teachers, the First Year Report, and the collection of Topic Outlines should allow other schools to take full advantage of our experience.

Health information and planning systems: The need for consolidation

by P. F. GROSS

University of Saskatchewan
Regina, Canada

INTRODUCTION

In a previous paper,[1] an evaluation was made of the state-of-the-art at that time in the use of the computer in hospitals, and as with any state-of-the-art evaluation, it serves only as a static focal point along a somewhat dynamic continuum. Various problems of using the computer in hospitals (and in the wider framework of health care), as well as some of the then-current issues in hospital automation, were also discussed briefly.

In this paper, it is proposed to focus attention on these issues and problems, with a view to advancing some curative remedies for what seem to be continuing ills, as well as advancing some thoughts on the preventative aspects for future health automation ills.

Finally, the research for this and the previous paper was carried out during visits over the past four years to projects in four countries (U.S.A., Canada, Britain and Australia) and during a continuing project, of which the writer was Project Director, involving eight hospitals in South Saskatchewan, Canada.

The continuing issues/problems of medical automation

Introduction

The five problems that appear to be most prevalent in medical automation (because of their continuing nature more than any other factor), are as follows:

a. the current level of interest of medical personnel in automation, as affected by (b) and (c)
b. the nature of present medical education and by the success or otherwise of computer system analyst/programmer education,
c. the state-of-technology with respect to computer hardware and software for medicine,
d. the medico/legal problems of computerized medical records,
e. the problems of rationalizing research projects into the development of large scale, on-line, real-time computer systems in hospitals (at present) and in health regions in the future.

These problems are discussed now in some detail.

Current level of medical personnel interest

At first glance, this supposed problem does not appear to exist, since a majority of the more successful projects in medical automation are at present headed up by personnel with medical qualifications: Collen at Kaiser Permanente,[2] Lindberg at University of Missouri,[3] Caceres at the U.S. Medical Systems Development Laboratory,[4] Vallbona at TIRR,[5] Lamson at UCLA,[6] and Barnett at MGH[7] are names familiar to the reader.

The above mentioned projects meet some of the generally accepted criteria for judging successes in the area of medical automation, which criteria might be:

- that the system is actually working, not proposed for implementation.
- that the modus operandi *and* the results are generally accepted by other workers in the field *and* the user organization.
- that the modus operandi is standalone, not in parallel with a manual system.
- that as pioneer ventures, they serve as beacons for future projects in similar areas, particularly to emphasize that the most successful projects are those where the problem is "... reasonably delineated, heavily oriented towards simple technology and involves individuals who are primarily oriented towards technology ..."[8]

The latter opinion of Barnett re the orientation primarily towards technology as a predictor of success in this area is very pertinent and does not necessarily preempt the "interest of medical personnel" as another necessary ingredient for success. Also, there are a large number of hospital projects aimed specifically at the *business* operations of the hospital that do not necessarily require full-time medical personnel support, the most impressive of which projects is probably Boston Children's Hospital Medical Centre.

With the foregoing sample of projects in *medical* automation that have been acclaimed by workers in and outside of the research area, the reader may well ask why the matter is even raised as a problem. As the literature on medical automation since 1964 indicates, there are at least 400 separate projects in progress as at 1969 in different parts of the world, and this number is growing, despite the number of projects that have not been successful. [Should the reader doubt the quoted figure, he is referred to a forthcoming book (Computers in Hospital and Health Systems) by the writer, in which book the bibliography lists over 400 *separate* projects]. With this number of on-going projects, the relative number of judged or potential successes does seem small. However, the most disturbing features of the large number of projects are:

- that they appear to duplicate work done earlier or similar work at present being undertaken by others.
- that only a very small proportion of the projects are operating *without* a parallel manual system, i.e., a majority of the projects are, at best, experimental models. It is claimed that the cause of this somewhat sporadic resistance by medical personnel to using the automated systems *in the clinical laboratory* (which is one of the more successful areas of automation endeavour in automatic data acquisition, storage and retrieval) is "... an array of unrelated instruments that can be more trouble than they are worth. If physicians are offered a good system ... you will find them quite willing to try it. . ."[9]

A more significant cause of the non-implementation of project output is the small numbers of medical personnel who are *actively* involving themselves in the research studies in the various institutions being studied. Dr. Barnett has indicated some possible causes for this situation when he states that "... the application of computer technology offers hope, but the realization of this hope in the near future [this statement was made in October, 1967] will require a much greater commitment than is presently true on the part of N.I.H. [U. S. National Institute of Health], the medical academic community and the health services community. *The critical weaknesses are*: (1) lack of imaginative and competent personnel to constitute the task force, and (2) an artificial separation both in N.I.H. and in the academic community between medical practice and medical administration and between research, development, implementation and service . . ."[10] (emphasis supplied).

Some of the factors mentioned so far are not related specifically to medical education, such as level of N.I.H. funding in the United States of medical automation research and the general quality of computer hardware and software. Both these problem areas will be discussed further on.

At this point, it is relevant to dwell on medical education as it affects the present and future development of computerized health information systems. In late 1967, Dickson noted that "... it is evident that basic knowledge in the area of physiological systems is not deep enough, nor broad enough to set up patient monitoring systems or intensive care units that will have the desired sophistication . . ."[10] Notwithstanding that this comment is directed towards the state-of-knowledge in a specific area of biomedical endeavor, a similar state of knowledge of medical automation and its problems exists *generally* in the medical profession. Since computer science is of only recent origin by comparison with medicine, it is safe to assume that a majority of the medical profession have never been required to take a formal course in the *elements* of medical automation, let alone the *specifics* of the subject.

The extent of possible remedies might be summarized briefly at this point:

1. It would seem that the computer education of the coming medical profession *must* commence at the undergraduate level as a required course. The progress being made by the University of Missouri, the University of Oklahoma, Baylor College of Medicine and the University of California at Los Angeles should serve as guidelines in this respect.
2. It would seem that continued efforts by professional bodies such as A.M.A. and A.H.A., through continuing short courses for medical personnel who have graduated, should be supported by government and other funds to enable a large number of courses to be given.
3. It would seem that universities might aid hospitals in their geographical area to commence in-house training within hospitals, regardless of whether the hospitals are training hospitals attached to the universities or not.
4. It would seem that the desirability of bringing together various disciplines to discuss modes of

attack on various problems of medical automation has been proved by conferences such as the international conference held in Washington in September, 1967,[8] by various national conferences such as the New York Academy of Sciences effort in January, 1968 and by a *restricted number* of conferences (such as FJCC and SJCC) where computer personnel attempt to resolve the continuing hardware/software problems that are discussed further on.

Summary: It is contended that the present level of interest of medical personnel in the automation of the medical environment is dangerously low. It is suggested that, as one of the factors that appear to correlate strongly with ultimate project success is medical personnel leadership, the medical profession should recognize the challenge of automation and upgrade medical education accordingly, a factor discussed in the next section.

Contemporary medical education and systems education

The second problem considered relevant to the present discernible plateau in medical automation research is the current lack of emphasis on the role of automation in the medical curriculum (but *not* the professional MPH or MHA curriculum), coupled with the general level of application system design expertise currently available. We shall discuss the two problems separately.

Medical education

The general lack of emphasis on the role of the computer in the medical curriculum suggests that the planners of such a curriculum have not yet fully realized that the computer will and is playing an important role in medical care. With a few notable exceptions [University of Missouri, University of Oklahoma, Baylor College of Medicine (Texas) and the University of California at Los Angeles], universities have not adapted to the challenge of the computer in medicine, the general exceptions being those universities involved directly with research either through research units within colleges of medicine or those colleges with affiliations with large health science centres.

With the consequent shortage of medical personnel with computer interests from their academic preparation, and with the present overcrowded syllabus in medical schools all over the globe, little relief can be seen for the present lack of medical personnel interest in automation, except indirectly through that small proportion of the medical profession who are biomedical research workers. One possible solution would seem to lie in the funding of several large medical academic operations, such as those listed above, in the hope that (a) the person with a particular interest in medical automation might be drawn to these centres; (b) the centres might be encouraged to publish even more widely the results of their work.

Another solution would appear to be the *limited* proliferation of reputable journals such as Computers and Biomedical Research (Academic Press) and Methods of Information in Medicine as media which attempt to span the medical and computer technology gap.

As the foregoing suggests, the reporting of on-going projects has been something less than objective or adequate, and through access to *accurate* project reports and articles in *medical* journals, the medical profession may be gently coerced into reading into subject matter which, at present, confuses rather than enlightens.

Systems education

Without considerably extending the length of this paper by paraphrasing the contents of the two recent papers,[11,12] it is contended that the dangerously low availability of expertise in the systems analysis and design of the hospital business or medical processes will present a serious, continuing hazard to the well-meaning hospital with intentions to automate. The medical/administrative interface of the modern hospital is difficult enough to bridge with *adequate* knowledge of both domains, but the current systems personnel shortage is forcing hospitals to accept people with *little or no* knowledge of those domains. There is nothing like a badly designed system to destroy confidence in *any* sector of industry, but the ever-vigilant opponents of automation in hospitals are particularly effective in their destruction of the reputations of hospital projects that are not meeting objectives.

Thus, there are two basic problems posed here:

- the availability of application systems designers generally
- the lack of *relevant* systems career preparations at the universities in particular, notwithstanding the excellence and relevance of degree programs in Information Systems at the University of Pennsylvania (MBA Information Systems Option), New York University School of Commerce (Computer Science major) and the University of Maryland Department of Systems Management. As has been indicated,[12] the available media for preparing the systems designers of the future are generally turning out a by-product not adequately prepared for

systems analysis/design in any sector, particularly the health sector.

The possible solutions to the education problem posed have been suggested,[12] and generally it appears that a massive up-grading of systems education at *all* levels is required. For the conference at which this paper is presented, this is a sobering thought that might be considered at this point.

Hospital administration education

It might be mentioned that the discussion thus far has not referred to the current trends in education for MPH and MHA programs in U.S. and Canadian universities, since it would seem that the professional MPH and MPA programs appear to have recognized the role of the computer in medicine, and no problem is seen to exist.

Summary: It would seem that the current state of medical and systems education does not reflect the need to impart more relevant education to medical personnel *and* to the systems analyst/designer attempting to match medical needs to existing computer hardware/software technology. This latter aspect is discussed now.

Computer hardware/software technology

At the present moment, the problems posed by the lack of, firstly, suitable input devices; secondly, by large low-cost bulk storage devices and, thirdly, by computer system reliability and cost are as serious as any of the other problems posed so far.

The input problem is causing other problems related to medical/ward personnel satisfaction with hospital information systems to date. It has long been recognized that what the computer industry needs is a reliable, low-cost, graphic display/keyboard terminal, and in the hospital, the requirements of such terminals have been summarized by Hofmann et al.[13]

Another paper[14] by Barnett and Greenes has summarized some of the relevant problems in achieving a working man-machine interface in computerized hospital system, as well as summarizing the 1968 state-of-input/output terminal technology. Essentially, the development of suitable terminal equipment is seen as being 4-5 years down a rather long road.

The availability of generalized software for the hospital information system has not led to a marked trend towards its implementation by the growing number of hospitals involved in the on-line, real-time mode of operation through time sharing. As other workers in other industries have found, generalized software is not easily modified for the particular requirements of those industries.

The gloomy picture of hardware and software inadequacies does *not*, of course, apply to all areas of hospital automation. In the clinical laboratory, the automation success stories far outweigh the failures for reasons that have much to do with Barnett's evaluation quoted earlier.[8] Essentially, successful laboratory automation projects have been marked by:

- restricted and relatively simple goals
- hardware that is relatively simple and not subject to technological failure
- an adequate feasibility study prior to selection of hardware and development of software systems, such a study being fully documented and its proposals accepted by all the parties concerned.

On the question of the need for a computer language that would enable physicians to use natural language input for data storage and retrieval,[14] the computer manufacturers have not yet indicated any formal interest in developing such a language. In the interests of consuming main core and processing time, it would seem that the current version of the JOSS language in use by the MGH investigators is certainly superior to most high level languages in use in other projects. Because the question of medical personnel leading the development of medical information systems would seem to be a preoccupation to this point, one might hope to see far more flexible languages than FORTRAN, PL/1 or COBOL soon becoming available to aid physician/investigators in developing better systems. Realistically, this development might be expected around 4–5 years up that long road being travelled by the computer terminal developers.

As at 1969, the ball has been long enough at the manufacturer's feet for some results to start appearing. It is perhaps time for Government to adequately fund a private development, given guidelines by workers in the field, perhaps through NIH or through SIGBIO.

Summary: The computer manufacturers have not adapted to the challenge of developing useable hardware and software for other than limited applications of which the laboratory is one. It could be expected that their comparative successes in this area might spur them on to the more pressing problems of hardware and software for the wider concept of the hospital information system in the various areas of *patient care*, allowing that the *patient accounting* area has been successfully automated in a number of on-going projects.

Medico-legal problems of the automated medical record

With considerable research effort being expended in the search for an acceptable computerized medical record suitable for medical research *and* for day-to-day operation of a hospital, there has been very little consideration of the legal aspects of such a record. However, because the public is now more willing to commence litigation against hospitals and practitioners who are negligent or who give the slightest cause to suspect negligence, it is with considerable interest that many hospital administrators watch:

- the rulings of Courts on the legality or otherwise of computerized medical records.
- the actions that hospitals take to maintain the security of medical records in an on-line, real-time environment.

In the first place, hospital records, generally speaking, constitute a source from which information may be obtained, but they are rarely evidence in themselves. At present, the legal process depends in most cases, and certainly in cases heard before juries, on oral testimony. Although there are many classes of documents which, once they have been identified, are admissible as evidence, this principle generally does not apply to records such as hospital records, which are really no more than notes made contemporaneously of matters observed at the time by various persons in the hospital.

Again, there have been many occasions when a doctor or a nurse, when his attention is directed to other parts of his own notes, is forced to admit that his original opinion, expressed in the witness box or in another section of his notes, is wrong. Because it is not possible to cross-examine a document, oral evidence still remains relevant and necessary.[15]

Again, questions as to signatures on and documentation of records, their maintenance and retention, accuracy of the recorded material (clearly the wrong diagnosis applied to a patient leading to wrong treatment, as perhaps might occur if extensive coding is required of the user, could give rise to a suit for damages), and the confidential nature of records have been discussed by Springer.[16]

Of particular interest is the protection of security of medical records in a time-shared system environment. The system has to ensure that the system *prevents* the wrong person from gaining access and that it *does not prevent* the right person from gaining access. A recent paper by Segall[17] is of considerable relevance here.

At this date, there is at least one ruling by an Indiana court on the admissibility of computerized evidence.[16] In view of the relevance of such a ruling on the doctrine *res ipsa loquitur* (which allows a plaintiff to put the burden of proving non-negligence on the defendant), it is repeated here. "As complicated mechanical devices of our modern age achieve perfection, and greater reliance upon them is justified, it follows that the doctrine (res ipsa loquitur) has a broader application . . ." (Ball Memorial Hospital vs. Freeman 196 N.E. 274 [Supreme Court, Indiana, 1964]).

Summary: It may be fortuitous that rapid advances in the development of the hospital information system concept have not proceeded rapidly, since a litigation-conscious population might have been well served by a computer system that periodically had hardware and/or software malfunctions. Since other industries using computers have had their share of system malfunctions leading to destruction of operating records or to the production of spurious cheques, it could be expected that hospitals will also have their share of automation woes.

The human life is one expensive commodity that far outweighs an operating record or a spurious cheque, and in the absence of any known court rulings on the admissibility of the computerized medical record, hospital administrators must seek wise counsel before implementing computer systems that have control of living processes, such as in patient monitoring or in the treatment of patients with severe drug allergies.

The reliability of the available hardware/software does *not*, at present, justify blind faith.

Problems of hospital automation rationalization

As one peruses the literature on research projects that are at present under development, it becomes apparent that many researchers are working in fields that are remarkably similar to those being researched by other workers. It would seem that the health authorities at federal level, by funding small and large projects, have created a situation where the small projects are useful only as demonstration projects, while the larger projects do not seem to be proceeding within the constraints of an overall plan of research for the nation.

The reason for this situation is that there is no overall plan for Hospital/Health Information System development, such as partly exists for Law Enforcement (NCIC System in U.S.A.) or for Education, because:

1. of difficulties in deciding what *criteria* should be used to assess the merits of alternative projects recommended for funding;

2. the attempts to develop the *concept* of the advanced "total" information system beyond the concept stage have failed badly to date, since, after many years of seeing project after project, cut back development to something far less grandiose than the "total" concept presupposes, the medical profession, the funding bodies and workers in the field have not yet seen an economical, workable system. The reasons for this continuing situation have been elaborated on elsewhere by Barnett and Greenes,[14] and will not be discussed here.

The most obvious result of this apparent setback is that hospital planners are now attempting, with considerable success, to develop automated systems for smaller functional areas of the hospital, and, at the same time, attempting to relate such development to the "total" information system framework, which has not been *adequately* documented since Dr. Fred Moore's pioneer work in 1962.[18] When one considers that the Lockheed company planned for at least four years *before* the first computer program was written for its production planning and control system, and only then commenced a five-year, five-stage development plan for 1965-69, it seems that our first efforts in large scale medical automation were defying precedent.

Notwithstanding the growing number of middle-sized and large hospitals which are now completely dependent upon computers for parts of their day-to-day operation, it should now be possible to conceive of a standard system design for a certain total bed capacity of a hospital or a group of hospitals, based on the past success stories in certain areas of hospital operation. The USNIH grant[19] to Lockheed to investigate the design of two separate information systems, one mechanical, the other computerized, was ostensibly aimed at rationalizing some of the on-going project development in hospital automation.

The basic requirement of such standardized systems is *transferability*, and, in this respect, the present non-standard mode of operation of hospitals across the world, their autonomy in many countries and the need to ensure that any standardization does not give any one manufacturer an advantage over his competitors, may all complicate the path to standardization. In the interests of efficiency and economy, it is doubtful whether such standardization can be validly deferred much longer. For one possible benefit that might be reaped by such a move, consider the decision that computer manufacturers might be forced to make with respect to the resources allocated to medical hardware/software development if they were faced with the formidable strength of large groups of hospitals demanding useable hardware/software according to specifications of a government sponsor. One has only to consider that the lack of a united front from CAI researchers has led to a slowing down of CAI hardware development by at least two of the manufacturers, who quite rightly demand that users tell them *specifically* what they want in hardware.

Summary: The need to rationalize system development in hospital automation has been realized by a number of researchers in the field, and government health authorities have made some tentative moves towards standardization.

It is suggested that the general lack of manufacturer zeal in developing *workable, reliable* hardware and software for the medical care operations of a hospital might be overcome by a united front from research groups in the field, supported by government directive.

Rationalization for health information systems

Introduction

In the above section, we discussed some of the problems that hospital automation is causing as at 1969. In this section, we discuss the concept of the Health Information System for a community, or for a state (or province) or for a nation. Lest this whole treatise fall apart on the basis of one definition, we define a Health Information System to be a set of procedures and processes aimed collectively at supplying health authority management at various decision making levels with information that is required by those managers to function effectively. Included in the procedures and processes might be combinations of manual and/or electro-mechanical and/or electronic data processing equipment. Depicted in its broadest sense, the schematic in *Figure 1* attempts to represent information flow, in one concept of such a system, through community, state (provincial) and national decision making levels. [*Figure 1* does not attempt to show all the relevant health care support units that should or could use such an information system at various levels.] Other broad concepts of what constitutes the health information system have been outlined by Bartscht[20] and by Flagle.[21,22] Since there have been very few attempts to assess inputs, outputs and mechanisms by which such a system might be planned, it is probably relevant to briefly outline some of the present on-going research that is producing information on the health of an individual, albeit in an uncoordinated fashion with respect to other research groups. These research projects outlined are not exhaustive, but representative of attempts to derive meaningful data that could serve as

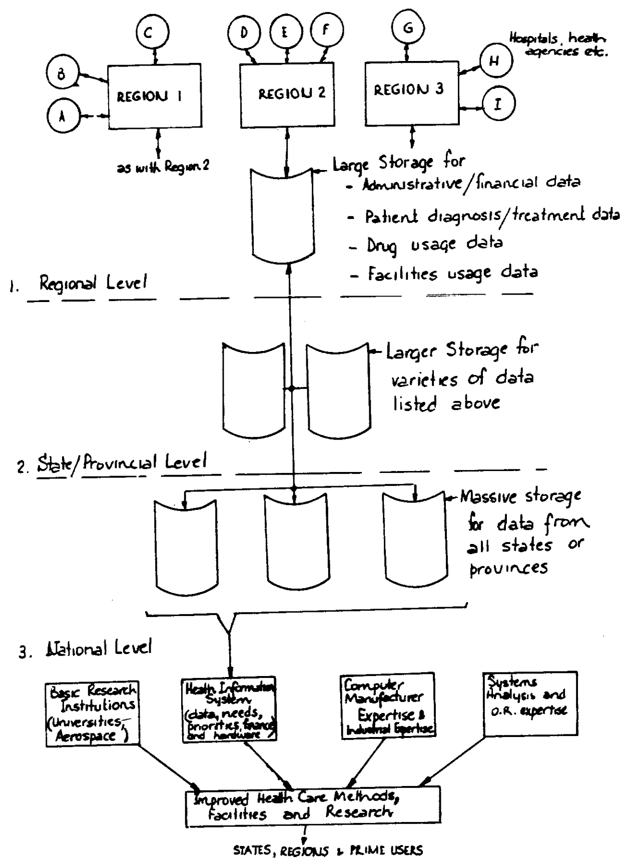

C
D
E
F
G
B
A
Hospitals, health agencies etc.
H
I
REGION 1
REGION 2
REGION 3
as with Region 2
Large Storage for
- Administrative/financial data
- Patient diagnosis/treatment data
- Drug usage data
- Facilities usage data
1. Regional Level
Larger Storage for varieties of data listed above
2. State/Provincial Level
Massive storage for data from all states or provinces
3. National Level
Basic Research Institutions (Universities, Aerospace)
Health Information System (data, needs, priorities, finance and hardware)
Computer Manufacturer Expertise & Industrial Expertise
Systems Analysis and O.R. expertise
Improved Health Care Methods, Facilities and Research
STATES, REGIONS & PRIME USERS

Figure 1—A schematic of a health informatio n system

input in a manner similar to that depicted in *Figure 1*, to large epidemiological studies of health communities.

The projects referred to are:

(i) the Kaiser Permanente Multiphasic Test Screening System at Oakland, California.
(ii) the Medical System Development Laboratory of the National Centre for Chronic Disease Control of the U.S. Public Health Service system for EKG recording and analysis.
(iii) the on-going record linkage studies in England, Maryland and New York in various health populations.
(iv) the Medical Audit Program and Professional Activity Study (PAS/MAP) of the CPHA.

The Kaiser Permanente multiphasic test screening system

At Kaiser Permanente Medical Centre in Oakland, California, a major research effort in the use of multiphasic test screening for preventative health care has been in progress since 1964. (For a detailed treatment of the form of testing and an assessment of the impact that such multiphasic test screening centres could have on health care, the reader is referred to published papers on the Kaiser Study.[23,24] At this moment, there is no other Medical Centre in North America that possesses the test screening facilities *and* the requisite data processing facilities employed by the Kaiser group, although the proceedings of the 1968 Conference/Workshop on Regional Medical Programs suggests that Tennessee, Missouri, Indiana and Connecticut will soon achieve working systems in various areas of those states. [The matter of whether the optimum mix of tests is in use at Kaiser or whether all the tests given are required will only be answered by a research effort lasting several years, and will not be discussed further here.] As the 1966 Senate hearings[23] and the above mentioned conference/workshop[25] reveal, several other entities in the U.S.A. are moving in the same direction as Kaiser, a situation that, at some time in the future, will require a linking of information flowing from test screening of large populations in different areas of the U.S.A. Despite the reservations of some physicians who see test screening with automated follow-ups as an encroachment on their professional sanctum sanctorum, there is sufficient reason to expect that only inadequate funding of such efforts, and perhaps a lack of improvement in computer speed and core storage, at reduced cost could halt any impetus arising from Kaiser's development at Oakland.

The Medical Systems Development Laboratory

The Medical Systems Development Laboratory has developed a computer system to measure values of heart rate, amplitudes and durations for wave-forms of the standard 12-lead ECG and to make an English-language translation of these measurements. During the past four years, over 75,000 ECG's have been processed, and it has been shown that, with limited direction from medical personnel, the system is useful in the detection, management and rehabilitation of persons with cardiovascular and associated diseases. Furthermore, the system decreases observer error and variation, reduces ECG costs and conserves scarce physician time for direction patient care.[4]

More significantly, a plan for a nationwide data pool of computer processed ECG's has been put into effect, with 35 investigative groups participating during the first year, with a combined annual output of 70,000 ECG's and an anticipated 200,000 in the second year. Also, it has been shown that it is feasible to process ECG's and spirograms from remote sites using conventional telephone circuits in an on-line, real-time mode, and also that outpatient care and emergency room services are possible.

As research into heart disease is a major concern at this moment, it is not difficult to imagine the effect of a nationwide data pool of information in a research effort, nor is it difficult to tie the impact of such research into the type of multiphasic screening approach in operation at Kaiser and other centres. The whole concept of preventative health care assumes new proportions, particularly if each health region, chosen by criteria as yet not universally agreed on, was financially capable of supporting the screening *and* data processing facilities required. The Tennessee Mid-South proposals for such a system are indicative of trends in this direction.[26]

Medical record linkage study

In various areas of the world, particularly in the U.S.A. and England, research into various aspects of comprehensive patient care as revealed by record linkage is proceeding, with the Oxford (England) studies[27] and the Maryland Psychiatric Case Register studies,[28] probably the best documented. Compared to the two previous research endeavours, this research is typically centered on a region, even though the New York studies[28] are centered on a smaller population. The fundamental problem experienced in the Italian studies,[27] the lack of data in a suitable format or the lack of data at all, has been met in all studies to date, and it is not difficult to postulate that the existence of data at municipal, regional and state level must precede any substantial research findings at a national level. The recent proposal of a research effort by the MRC in England[30] is fundamentally the first step in the right direction.

The PAS/MAP system

At a national level, a research effort enabling a medical audit of over 1240 hospitals annually discharging nearly 10 million patients across the U.S.A. (and other parts of the world) has been in progress for some years through the facilities of the Commission on Professional and Hospital Activities, which is an educational and scientific organization sponsored by the American College of Physicians, the American College of Surgeons, the American Hospital Association and the Southwestern Michigan Hospital Association.

Inherent in this system of medical audit (called the PAS/MAP system) are two basic criteria:

- the establishment of a standard relevant to the different treatments being given in hospitals
- accurate reporting of the actual treatment given to the patient during his stay.

Since the participating hospitals, in reporting their monthly statistics to the central computer facility in Michigan, are forced to code their treatments into a fixed format reporting sheet, some measure of standardization is present, but it would be unrealistic to think that *all* relevant information on a patient treatment can be included on any fixed format coding sheet. For this reason, while the PAS/MAP is a first step towards a nationwide pool of information (since 1240 hospitals is not the full complement of hospitals in U.S.A.), there appears to be a real need to incorporate systems such as PAS/MAP (or the Pittsburgh HUP system) within a national plan of health care funded by the national government, and with *all* health care units, down to the smallest nursing home or hospital, participating. The possibilities for the rational disposition of the nation's funds for new or improved health facilities in the smallest region, as distinct from the ad hoc methods in vogue in most nations of the world, are apparent here, as are the possibilities of comparison of cost and effectiveness of patient care in different regions. The development of realistic simulation models of health care for the nation looms as a possible output (given sufficient research into the validity and extent of the input parameters of the health care process), enabling long term planning of expensive health care facilities to be undertaken. Some results reported by Centner et al.[3] and by Ryan and Dillard[32] are significant steps in a move towards the Total Health System Model, as is the steady progress being made in econometric modelling of the health sector by Feldstein,[33] Yett and Mann[34] and others.

Problems of health information system research rationalization

Introduction

As Elam has noted, "... our problem [in America] is not that we lack technology; we have enough to frighten us all. It is not that we lack a concept of comprehensive health care or that there have not been isolated attempts to practice it. *What we do not have is a marriage of technology and comprehensive personal health care, and an assessment of the results* (emphasis supplied). . . ."[26]

As has been suggested earlier, the success of Collen's work at Kaiser Permanente (and Jungner's work in Sweden) has led to proposals for similar systems in regional medical programs now developing as a result of Public Law 89-239. One of the most significant of these programs is the Tennessee Mid-South Regional Medical Program described by Elam,[26] where researchers are attempting to determine whether comprehensive, family-oriented health care in a neighborhood health centre coordinated with an automated multiphasic screening laboratory will result in improved mortality, morbidity, health service utilization and health attitudes amongst a population. The determination of whether such a program reduces the cost of medical care *and*, at the same time, whether it improves and restores the family unit, is also proposed.

There are many other on-going developments across the U.S.A. for similar studies to implement the aims of Public Law 89-239. Since these proposals involve the use of health care facilities including hospitals, what are the likely problem areas that might forestall the achievements expected for comprehensive health care using automated health information systems?

The problem areas

The major problem areas are those already discussed in the paper to this point, and essentially they are related to resources—manpower, money, machinery—and to the method of attack.

Manpower – As has been suggested, manpower problems will possibly appear in varying proportions of:

- *medical personnel* with little or no knowledge of automation, even at such a level as would render the automation of screening programs more meaningful to them.
- *systems analyst/programmers* with little or no knowledge of medical systems.

If we cannot solve the manpower problems that exist in current isolated hospital projects, there would seem to be little chance of *adequately* staffing regional developments of health information systems, particularly if all systems require staffing of the order suggested below.

Costs – The financial crunch that has hit all sectors of the economy will worsen the funding situation for medical research programs such as envisioned by Tennessee, Connecticut and other forward-looking states. It is worthwhile recording the actual level of cost of automation in hospitals that have grouped together by studying the Indianapolis Hospital Association report[35] on estimated expenditures, and attempt to relate N repetitions of such a cost to a nationwide hookup of health care systems.

For a computer system capable of supporting thirteen area hospitals (6000 beds plus) and five health agencies, the proposal estimated that

- 79 people would be required in the period 1968-71 to administrate, analyze, design, program and operate the system;
- the system, if commenced in 1966, would cost \$9.4 million by the time all aspects of the system were operating in 1971, at which time the operating cost would be \$2.4 million per annum.

It is emphasized that the Indianapolis studies are just one of a number of shared hospital computer studies ongoing at this moment in the U.S.A. Assuming that such hardware and people support is capable of performing the data processing tasks of an automated (in the equipment sense) multiphasic test screening laboratory for a region containing hospitals and health agencies, the cost of renting automatic, multichannel analyzers plus the cost of reagents is estimated at being in excess of \$0.20 per test, without allowance for labor, space and allied costs.[36]

Hardware – As has been indicated earlier, the hardware required for collecting and processing multiphasic test screening data is already in use at Oakland, but when many hospitals are linked together in remote access to a central facility, the size, speed and cost of direct access storage, plus the input terminal problem posed earlier, would still be present.

Thus, certain elements of a comprehensive health care program can already be automated, while others will depend very much on manufacturer motivation before worthwhile results will appear.

In essence, it would appear that existing systems staffing shortages, coupled with the current limited level of computer expertise in the medical profession and the need to pressure manufacturers to provide reliable working hardware for a large-scale remote access, time shared hospital operation, *should* lead to the consolidation of all that is good in on-going hospital automation projects into one or two projects supporting a comprehensive health care system.

This consolidation of expertise, funding and pressure on manufacturers by the federal government *and* by groups such as SIGBIO, would result in the one or two projects being adequately funded and supported with staff and equipment commensurate with the system needs. It would not necessarily mean the future cutback of funds on all on-going projects, but would probably mean that the rate of increase of *new* projects might decrease.

At this moment, it is doubtful if subsequent evaluation of the efficiency of such consideration would not indicate

- better use of the *available* human resources (systems and medical)
- a favourable benefit/cost ratio compared to a proliferation of smaller projects
- that manufacturers reacted appropriately to pressure to design equipment for such systems.

CONCLUSION

Some of the more significant problems and issues in medical automation have been discussed, and related to each other.

It is suggested that current human resource shortages, together with the costs of and working efficiency of large-scale computer hardware facilities, should lead to a consolidation or rationalization of on-going research into medical automation.

The effects of automation within a comprehensive health care system might really be felt if such rationalization led to more efficient use of all resources, financial and others.

ACKNOWLEDGMENTS

The critical advice of Drs. Barnett, Caceres, Collen, Lamson and Vallbona through their correspondence, is greatly appreciated, as is the typing expertise of Miss S. Wenger, Secretary of the Faculty of Administration, University of Saskatchewan, Regina.

BIBLIOGRAPHY

1 P F GROSS
The computer in health care: Needed—A plan for development
World Hospitals Vol 4 : 4 189–200 October 1968

2 M F COLLEN
The multitest laboratory in heal h care of the fu'ure
Hospitals JAHA 119–125 May 1967

3 D LINDBERG
Computers and medical care
Springfield Illinois Charles C. Thomas 1968

4 C CACERES
In: Computers, electrocardiography and public health—A report of recent studies
PHS Publication 1644 USPHS US Department of Health Education and Welfare June 1967

5 C VALLBONA et al
An on-line computer system for a rehabilia'tion hospital
Methods of Information in Medicine Vol 7 : 1
31–39 January 1968

6 B G LAMSON
Data processing in a medical centre—Progress report 1966
Department of Pathology School of Medicine Center for the Health Sciences University of California Los Angeles 1966

7 *Massachusetts General Hospital Computer Project—Progress Report May 1966 to September 1967*
MGH Memorandum 10 October 1967

8 Personal communication from Dr. O. Barnett of Massachusetts General Hospital 28.1.69

9 N LINDGREN
Future goals of engineering in biology and medicine
IEEE Spectrum 93–100 November 1967

10 G O BARNETT H J SUKENIK
Hospital information systems
Invited paper given at NJH Conference
Future Goals for Engineering Biology and Medicine
September 9 1967 reproduced in MGH Memorandum 10
October 19 1967—see reference 7

11 P F GROSS
Crisis in system education
Canadian University 3:4 30–33, 56 June 1968

12 P F GROSS
Systems education—Needed: A reassessment and upgrading
Faculty of Administration University of Saskatchewan (Regina) January 1969 (mimeo)

13 P B HOFMAN W A GOUVEIA G O BARNETT
Computers: great future perilous present
Modern Hospital July 1968

14 G O BARNETT R A GREENES
Interface aspects of a hospital information system
Paper presented at New York Academy of Sciences Conference on Use of Data Mechanisation and Computers in Clinical Medicine January 15–17 1968

15 G SAMUELS
Medico-legal aspects of clinical records
National Hospital Vol 11 : 5 9–18 February 1967

16 E SPRINGER
You, the computer and the law
Medical Record News 15 February 1965

17 J SEGALL
Legal implications of automated hospital information systems
Paper presented at the Conference on Advances in the Applications of Computers in Hospital Management University of Michigan Engineering Summer Conference May 15–19 1967

18 F MOORE
Health information systems: Volumes I and II
University of Southern California School of Medicine 1962
(Available from USC Bookstore 2025 Zonal Avenue Los Angeles 33)

19 *New contracts*
Computers and Automation 50 September 1967

20 K BARTSCHT R JELINEK
A management information and control system for health services
Paper presented at University of Michigan Engineering Summer Conference on Advances in the Applications of Computers in Hospital Management May 15–19 1967

21 C FLAGLE
Criteria for evaluation of health service information systems
Paper presented at University of Michigan Engineering Summer Conference on Advances in the Applications of Computers in Hospital Management May 15–19 1967

22 C FLAGLE
Implications of health service information systems
Paper presented at University of Michigan Engineering Summer Conference on Advances in the Applications of Computers in Hospital Management May 15–19 1967

23 *Detection and prevention of chronic disease utilizing multiphasic health screening techniques*
Hearings from the Subcommittee on Health of the Elderly of a Special Committee on Aging—U.S. Senate 49th Congress Second Session September 20–22 1966 U.S. Government Printing Office

24 M COLLEN
The multitest laboratory in health care of the future
Hospitals JAHA 119–125 May 1 1967

25 *Proceedings of conference/workshop on regional medical programs January 17–19, 1968, Volumes I and II*
National Institute of Health US Public Health Service
US Department of HEW Washington DC

26 L ELAM et al
Health evaluation studies utilizing a multiphasic screening centre operating with a comprehensive health care program for persons in an urban poverty area
In 25 Volume II 1–4

27 *Mathematics and computer science in biology and medicine*
Proceedings of a Conference held by [U.K.] Medical Research Council in association with the Health Department Oxford July 1964

28 W PHILLIPS
Record linkage for a chronic disease register
Paper presented at the International Symposium on Medical Record Linkage Oxford England July 1967

29 D NITZBURG
The methodology of computer linkage of health and vital records
Reprint from Social Statistics Section—Proceedings of Amer Stat Assoc 1965

30 *U.K. council proposes medical data bank*
Datamation 105–106 September 1967

31 S CENTNER et al
Computer simulations to assist rational resources allocation in health science education
Health Science Functional Planning Unit
University of Toronto November 1967

32 J RYAN C DILLARD
A computer model of the total emergency care system
Paper presented at 6th Annual Southeastern Regional Meeting of ACM and National Meeting of Biomedical Computing University of North Carolina June 15–17 1967

33 M FELDSTEIN
Economic analysis for health service efficiency
Amsterdam North-Holland Publishing Company 1967

34 J K MANN D E YETT
The analysis of hospital costs: A review article
The Journal of Business Vol 41 : 2 191–202 April 1968

35 *EDP implementation planning study, technical report, Indianapolis hospital development*
Information Science Associates Cherry Hill New Jersey 1966

36 D SELIGSON
Provision of optimum chemical laboratory services for 3,000,000 people
In Proceedings of Conference/Workshop on Regional Medical Program op cit 4–6

Computer assisted instruction in the diagnosis of cardiac arrhythmias*

by E. J. BATTERSBY

Vanderbilt University School of Medicine
Nashville, Tennessee

INTRODUCTION

Disorders of heart rhythm, known as cardiac arrhythmias, are most accurately diagnosed from selected electrocardiographic (EKG) tracings. Their interpretation is necessary for the correct treatment, sometimes on an emergency basis, of a large number of patients with heart disease. Analysis of the contour and sequencing of component waveforms taken with the knowledge of the set of defined electrophysiologic mechanisms allows categorization and in most instances specific identification of the rhythm. Most of these diagnoses depend upon the specification of the site or sites of electrical impulse generation in the heart and its subsequent spread through specialized conduction pathways. The logical process leading to the diagnosis is inferential because the tissues involved in impulse generation and specialized conduction are of such small mass as to have no direct electrical representation on standard EKG leads. Only the electrical activation of larger masses of cardiac tissue as a resultant of those fundamental mechanistic events produces the component waveforms of the clinical electrocardiogram.

It is the purpose of this paper to describe a digital computer simulation of the events that are the determinants of cardiac rhythm and the resultant activation of those portions of the heart that result in the generation of the clinical electrocardiogram. The simulation is intended to assist instruction in the inferential process of arrhythmia diagnosis. A number of analog devices with a restricted repertoire of rhythm disorders are available but to the author's knowledge the system described herein is the first allowing programming of the basic mechanism of rhythm determination.

To be useful in training medical and paramedical personnel in this important subset of clinical electrocardiography several goals must be met. The language of coding used to change basic mechanisms must be minimal and in phrases acceptable to a student of electrocardiography. The simulation should be based upon accepted mechanistic behavior of well defined electrophysiologic components and not merely a set of arbitrary "function generators". The output of the simulation should be an analog signal that appears to the student as a bona fide EKG lead as would be recorded from a patient in real time. Changes in the behavior of the underlying mechanism during the course of a rhythm strip generation must be implementable with display of the resultant without any break in continuity of the analog output across the point of change of mechanism. Not necessary but desirable from the pedagogic point of view would be expansion of the time base of the output to provide a "slow motion" electrocardiogram for oscillographic viewing.

Two versions of this program have been implemented. One is in Fortran IV and was written to define the logic in an easily communicable form and to yield a detailed sequential listing of events to provide interactions of events. The version described herein in detail is written in assembler language for an SCC-650/2 digital computer with a 12 bit word length and 8 K of memory. During the input of parameters, each command is evaluated for syntax and the appropriate parameters inserted into the model. When a command is given to execute the model, a poll of all the functional modules is conducted at a simulation interval of 1/200th of a second. As particular portions of the simulated heart are fired, those contributing to the EKG appear

* This publication is supported by the following Grants:
US, PHS Career Development Award, Grant #1-K3-HE-10, 130
US, PHS Program Project Research Grant #HEO 8195
US, PHS Cardiovascular Training Grant #5 T2 HE 5025

as a string of numbers stored in memory. The form of each type of waveform is available in a pre-defined list. At the time of display the string is passed to the D-A converter for display.

Operational logic

Pacemakers

After electrical discharge a pacemaker region undergoes a predictable recovery cycle leading to spontaneous discharge if uninterpreted. This type of periodic behavior is the mechanism that allows the heart to produce its own intrinsic repetition cycle. During the initial phase of recovery the pacemaker is said to be refractory and is insensitive to waves of depolarization travelling in adjacent tissue. After this refractory period local discharge may "enter" the pacemaker region and discharge it causing it to return to the beginning of the recovery cycle. This is known as pacemaker resetting and pacemakers protected from this discharge from without are known as parasystolic. When repeatedly reset from without the duration of the recovery period from onset to spontaneous discharge may be lengthened and when subsequently released from such suppression will gradually return to its basic period over the course of several beats. This escape phenomenon is termed warming. For the description of these kinds of pacemaker behavior five specifications are necessary. The length of the refractory period is set as fixed data. The pacemaker command in the simulation language supplies an identification number, a number describing the length of the spontaneous discharge cycle and two Boolean variables specifying whether or not the parasystolic and warming phenomena are in effect. Thus, with blanks as separators a typical command appears as follows: P2 75 W; which specifies that pacemaker two has a cycle length of 0.75 seconds and exhibits warming. The absence of a P in the third argument means that the parasystolic bit is not altered. The location of pacemaker two in the geometry of the heart will be considered in the Interaction section.

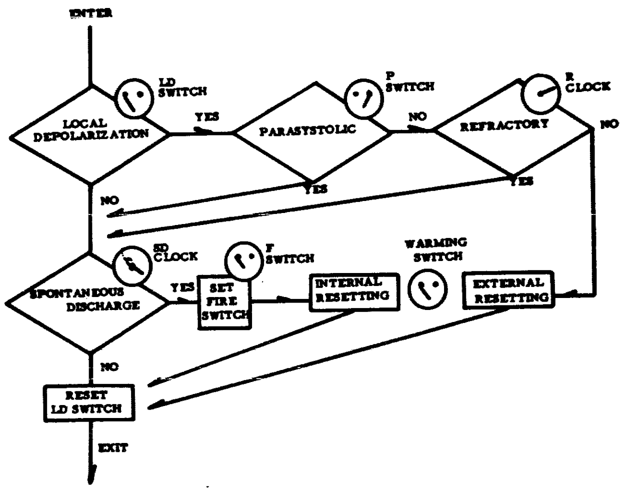


Figure 1—Pacemaker logic flow
This is the chain of tests performed on each simulation cycle of a pacemaker. Further description is in text

Figure 1 describes the logical steps for a pacemaker during a single cycle of the simulation. Excepting some details it may be read as follows:

IF	local depolarization has occurred and the pacemaker is neither parasystolic nor still refractory
THEN	reset the clocks and GO TO EXIT
ELSE IF	if the discharge clock countdown is complete
THEN	set the indicator to note attempt to fire adjacent tissue and reset the clocks
EXIT	reset the local depolarization counter.

Specialized conducting tissue

The most important functional elements of the specialized conduction system join the two upper chambers of the heart, the atria, with the two lower chambers, the ventricles, as a conductor of impulses in either direction. As there is ample physiologic evidence that alterations in conduction may occur at several levels in this longitudinally distributed pathway it is simulated by a chain of similar modules that may be separately controlled by parameter modifications and which interact with adjacent elements. This pathway, because of its relatively slow conduction velocities provides a delay in normal circumstances between the electrical activation of the upper and lower cardiac chambers. In abnormal situations, it may produce an unusually long delay or fail to conduct. This transmission line, known by several names to electrocardiographers, will be called the junctional tissue in this paper.

A typical recovery pattern for this junctional tissue may be divided into four phases in terms of its behavior in transmitting impulses. For an initial period of time following discharge the tissue is said to be refractory meaning it will not respond in any way to the arrival of an impulse. In the second phase, impulses may be partially conducted in the region but will not in turn execute the next succeeding segment. Sequential conduction will fail as in phase 1 but the partial discharge of the segment will result in resetting of the recovery sequence back in time. The effect upon subsequent conduction behavior for impulses arriving later may be different. In phases 3 and 4 sequential conduction does occur. In phase 4 after full recovery the conduction delay is a fixed minimal time. In phase 3, still incompletely recovered, the segment conducts with a delay that is a nonlinear function of the time still necessary to accomplish full recovery.

A typical command specifying the behavior of such a segment appears as follows: J3 100 40 10; denoting that junctional element three requires 1.0 second for full repolarization of which 0.4 seconds is the length of phase 3 and 0.1 seconds is the length of phase 2. Phase 1 is the remainder and the fixed delay of phase 4 is fixed data in the program.

Figure 2 shows in schematic form the logic of this type of conduction module. Boolean "up" and "down" are specified so as to indicate the direction of conduction. Thus, when an impulse is passed to such a segment simulator a directional indicator is passed which will determine the subsequent passing sequence if the segment conducts. The initial test is to determine if the impulse pass bit is set. If it is the recovery clock is checked for phase. If in phase 3 or 4 the delay is calculated and used to set the impulse clock and the directional flags are set. If in phase 2 the recovery clock is partially reset and no impulse action is taken. No action is taken if the impulse switch is off on entry or if it is on and the segment is in phase 1. Then the clocks are advanced and at a cycle when the impulse clock countdown is completed an impulse will be passed in the direction prescribed by the directional flags.

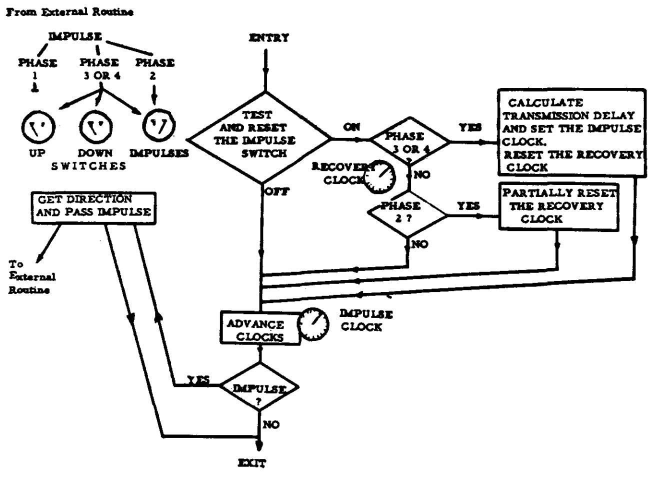


Figure 2—Junctional element logic flow
This is the chain of tests performed on each simulation cycle of a Junctional Element. External Routines refer to similar logical modules of adjacent Junctional Elements. Further description in text

Certain abnormal rhythms are attributed by electrophysiologists to reentry paths. These are conduction pathways isolated in such a way as to be protected from complete discharge by a wave of electrical depolarization passing through their region. Rather they are shielded and slowly conducting conduction limbs whose cycle is initiated by a passing impulse and whose transit time is slow enough that when they exit locally or in nearby tissue they may initiate another impulse propagation. This cycle may then recur or be extinguished on the next pass. These may behave in certain instances as "rebound pacemakers" and in other instances to return impulses back through a strip of specialized conduction tissue in the opposite direction from which it entered.

The reentry command has the form R4 20 3; in which the first specification identifies the reentry path, the second specifies a delay of 0.20 seconds, and the third argument allows a single digit of zero through eight that sets up "chances-in eight" of the event occurring at any particular trial by reference to a random number generator.

This type of logic may also be used to specify bypass pathways of conduction with fixed delays that are invoked to explain certain types of abnormal impulse propagation.

Interaction

The communication among the elements described above and the larger masses of cardiac tissue whose electrical activity is seen in the standard electrocardiogram depends upon their relative geometric disposition as well as their functional properties. Figure 3 shows the spatial relations of the elements simulated in the model. The principal potential pathways of conduction are indicated by directed arrowheads. Programmed reentry loop paths are omitted for simplicity.

The spread of excitation in one of the elements of the atria or ventricles has several consequences. It may reset the spontaneous depolarization cycle in an adjacent pacemaker. It may after a period of time initiate activation of an adjacent element. It will supply from a table amplitude information to the D-A converter routine for display. Since the direction of depolarization of the portions of the atria may be either upward or downward and the direction of the ventricular septum rightward or leftward, and since this will affect the resultant waveform, these possibilities are accounted for in the program and the waveform element tables. The waveform of repolarization of ventricular elements is seen on the surface electro-

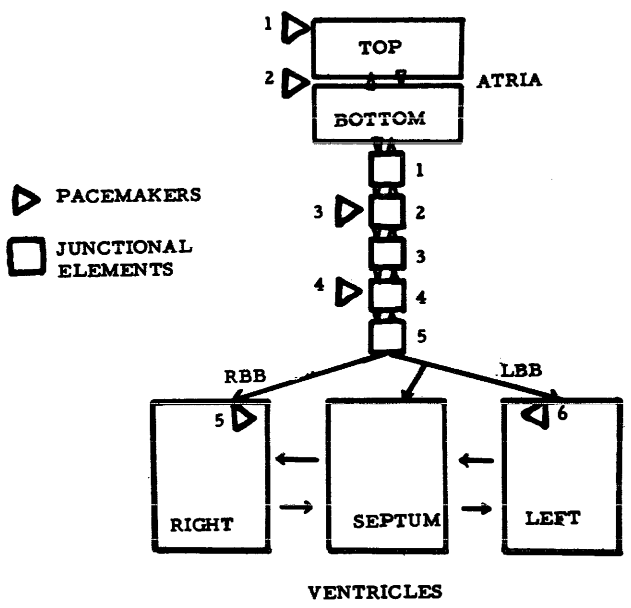


Figure 3—Disposition of the simulated portions of the heart. Note that a string of junctional elements connects the atria (the upper portion of the heart) with the ventricles (the lower portions of the heart). The septum is the wall separating the two ventricles

cardiogram and is supplied to the display modules.

Note that transmission from the fifth junctional element downward is by two pathways, one to the right ventricle called the right bundle branch and one to the left ventricle and to the left side of the septum called the left bundle branch. Normally, conduction down this latter pathway will cause left to right depolarization of the septum. Since certain changes in the EKG waveform will result from deletion of either or both of these pathways, such an option is included in the programming language.

Simulation language

An interpretive program for the simulation language was written in assembler language for an SCC-650/2 digital computer with 8 K of memory and a 2 microsecond cycle time. This allows teletype input of commands and viewing of the output immediately after construction of a rhythm strip either on an oscilloscope face or a strip chart. So as to be able to execute previously tested programs in the simulation language, command strings can be input from paper tape. For the remainder of this discussion keyboard input will be assumed.

Commands in the language may be divided into three types:

1. Those that establish electrocardiographic parameters.
2. Those concerned with storage and display of the rhythm strips.
3. Exits out of the command program and input device selectors.

Each command is preceded by a colon supplied by the program and indicates readiness to accept the command. Each command is terminated by a semicolon and at any time prior to the entry of the semicolon terminator, a colon may be used to delete the command. Comments fields are admitted. A command is syntactically evaluated after the entry of the semicoln terminator and before execution. Syntax error messages are generated and allow immediate reentry of a correct command.

Type I commands that determine EKG parameters are those described under pacemakers, junctional elements and reentry paths and four others. Time of the simulated rhythm strip in integer seconds from one to thirty may be specified by the command:

TIME 8 SECONDS; A command exists establishing normal parameters so that the rhythm will be produced by pacemaker one and transmitted without conduction delay or disturbance resulting in normal sinus rhythm in the electrocardiographer's language. This is of the form:

NSR; and is useful in establishing baseline initial conditions. Lead selection commands allow the operator to view either of three lead displays. Lead II is the standard body surface lead used in rhythm diagnosis and is established by the command:

LEAD II; In addition, a lead taken in special circumstances from within one of the atria in which the atrial deflections may be more clearly discerned may be designated. Finally, a lead not available to the clinical electrocardiographer is generated which shows by spikes the passage of conduction through any or all of the junctional elements. The command:

BBBL; establishes block of the left bundle branch and commands are available to create or remove block of either the left or right bundle branch.

Storage and display commands allow storage and retrieval for display of generated rhythm strips. The command:

UPDATE; states that the operator is ready to begin generation of a new string of rhythm strips. This simply resets a memory pointer. If it is desired to have the rhythm strip displayed on an output device during generation the command:

VIEW; is used. This will generate the segment according to the functional element parameters then present. The command:

ENTER; will do exactly the same but without display. Display of the sequence of arrhythmias generated since the last update command without break in continuity is accomplished by the command:

DISPLAY;

Other commands are not formally part of the language but used in the present implementation to transfer control to a debugging program and a device flip-flop that transfer control to the paper-tape reader when executed on the teletype and vice versa.

The coding of the simulation language is minimal and is in terms readily accepted by the student of electrocardiography. More detailed description of the command structure is specified in a manual defining the language and the presentation here is abstracted from that source.

RESULTS

By definition, rhythm changes that are produced by other than the mechanisms defined in the model cannot be produced. They are few and in general not amenable to the usual type of analysis that this simulation is expected to assist in teaching. While they might be included in the repertoire by supplying a list describing them this would not increase the usefulness of the simulator as a pedagogic tool.

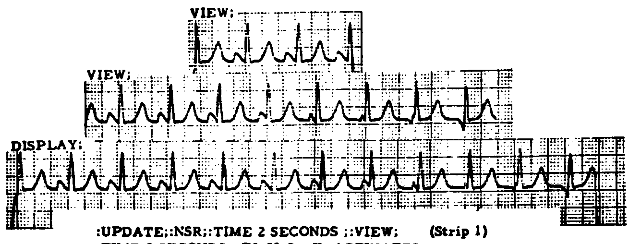
A wide variety of important rhythm disorders can be produced yielding a noise free signal and the opportunity to view special leads that is not usually available in the clinical setting. The underlying mechanisms are clearly defined by the operator's input commands. Some examples of the tracings produced by the machine will be considered to illustrate certain features of the system.

Figure 4 illustrates how the display command may be used to portray as a continuous tracing two segments that were sequentially created with a change in mechanism between them. The test below the records is the nonerating program. Strip 1 is a two second run of normal rhythm. Then the rate of pacemaker three (upper nodal focus to the electrocardiographer) was increased and a five second record created. Then, by use of the display command the entire seven seconds are seen as continuous record. The point of fusion is indicated by the triangle.

The interval of time between the onset of the wave associated with the firing of the atria and that associated with firing of the ventricle is usually constant in normal situations. If the period of the pacemaker firing the atria is shorter than the recovery period of a junctional segment the arrhythmia seen in Figure 5 may result. Because of incomplete recovery this interval is progressively longer until conduction is blocked; recovery is allowed and the cycle repeats.

Note also the atrial waveform obscured in the portion of the record indicated by the triangle. An experienced cardiologist would recognize this. The short strip below shows the appearance of the special atrial lead in this circumstance showing the clear presentation of the atrial wave obscured in the upper tracing which would alert the student to the obscured wave in the standard tracing.

Figure 6 shows the usefulness of the other lead se-



```
:UPDATE;:NSR;:TIME 2 SECONDS ;:VIEW;      (Strip 1)
:TIME 5 SECONDS ;:P3 55 O ;:Y; ACTIVATES
THE UPPER NODAL FOCUS AT A RATE
SLIGHTLY GREATER THAN THE SINUS RATE
OF 100 *:VIEW;                            (Strip 2)
:DISPLAY ;                                (Strip 3)
:
```

Figre 4—Demonstration of fusing of two separately generated strips. See text

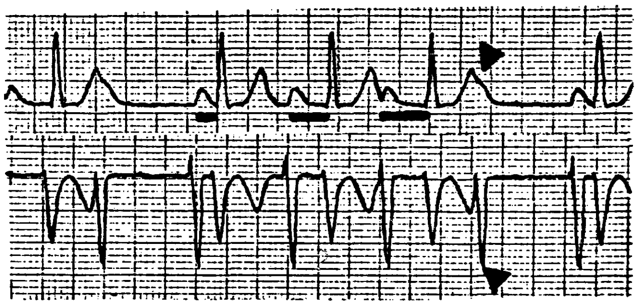


Figure 5—Demonstration of the use of the simulated intra-atrial lead (bottom strip). See text

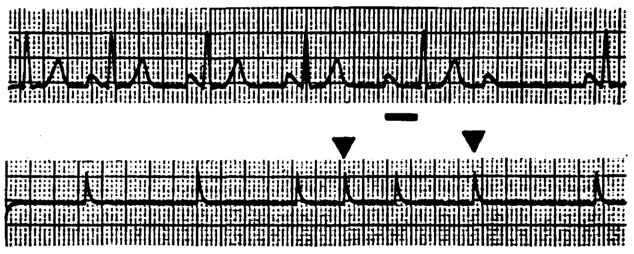


Figure 6—Demonstration of a lead marking activation of one of the junctional elements (bottom strip). The effect of the nonconducted impulses marked by triangles on subsequent conduction is seen. This is an instance of so-called concealed conduction

lection option. Here the lower strip shows activation of one of the junctional elements indicated by a spike. At the two instances indicated by the triangles the junctional element is activated by an adjacent pacemaker while in phase 2 of its recovery cycle. It does not propagate the impulse but the effect on subsequent conduction is seen. In the first case the interval between atrial and ventricular activation is prolonged (indicated by the bar) and in the second instance the spread of the impulse is blocked. This is an example of what a cardiologist would refer to as "concealed conduction."

Strip chart recordings produced by the system have been used by the author as instructional material for continuing education courses for physicians and for students in the medical school curriculum. On line oscilloscopic displays have been used in the education of Coronary Care Unit nurses. At present, there is no proof that this approach that synthesizes rhythm abnormalities from underlying mechanisms will allow students to develop the analytic ability that undergirds arrhythmia diagnosis. A classroom is currently being supplied with teletype and output devices and a proposal to train half a medical school class using the system and the other half in the standard manner is being entertained. It is hoped that this will yield information as to whether this experiment in computer assisted instruction is in fact superior to standard classroom instruction. The emphasis has been thus far on the instructor as user, rather than the student. In the course of the planned evaluation it is expected that the student will be able to interact with the simulator on an individual basis.

Hospital automation: Something more than a computer

by WALTER L. BENNETT, CHARLES F. STROEBEL and BERNARD C. GLUECK, Jr.

The Institute of Living
Hartford, Connecticut

INTRODUCTION

The introduction of computer techniques into the hospital environment offers an exceptional opportunity to reassess traditions and procedures developed over the years of a non-automated era.[1] However, there is an apparent danger that computer applications evolving in many hospitals tend to perpetuate the stereotyped roles of departments and personnel confined within traditional organizational boundaries. Their primary emphasis on conventional business or other specialized areas serves to sustain long standing and often outmoded rituals and procedures, imbuing them with the aura of modern automation. Such stereotype can be avoided by centering design of the computer system on the patient and his care as the crucial basic unit, thereby optimally meeting the needs of both patient and staff.

This approach provides "something more" than mere automation; it reduces communication barriers by integrating and focusing the efforts of the administration and a diverse staff on optimizing the care of the patient; it encourages a redefinition of traditional duties and roles to minimize the functioning of human beings at repetitive, mechanical tasks which can easily proliferate with an unimaginative installation of data processing equipment; it encourages a fuller and more flexible version of the ultimate capabilities of computer technology in providing complete medical care.

This paper reviews and generalizes on the experience gained in the application of the patient centered concept to the implementation of a computer based information system in a 400 bed private psychiatric hospital, using an IBM 1440 computer and 12 Bunker Ramo cathode ray tube terminals for on-line service eight hours a day for the past two years.

The proliferation of computers in hospitals has been extensive. A survey made in the Fall of 1962[2] showed that only 39 hospitals were using computers. Four years later, January 1966, the American Hospital Association reported in their annual survey of "Hospitals Accepted for Registration" that 586 hospitals either had their own computer or received computer services through a service bureau. From the Fall of 1965 to May, 1968, ECHO, a national organization of hospital personnel engaged in the installation of computer systems, experienced a membership increase from 74 representatives of 49 instutitions to 457 representing 241 private institutions plus numerous state and federal hospitals systems.

A review of current periodical literature and ECHO membership applications describing the status of hospitals' automation programs reveal that the majority of operating systems emphasize patient billing, accounts payable, general ledger, payroll, inventory, and statistical reporting programs. A smaller number report the development of computer applications serving the research department or the clinical laboratory.

In the vast majority of hospitals, it appears that the implementation of computer services is under the direction of the comptroller, a comparable administrative officer to whom the data processing personnel report, a research scientist, or a laboratory technician. In relatively few instances is the responsibility for an automation program vested at an administrative level where the joint requirements of the whole hospital—clinical, research, business affairs—are seen in their proper perspective and can be effectively incorporated into an overall plan.

Considering those in whom control of computer programs is commonly vested, this concentration of attention and resources in the more conventional and better understood uses of the computer for business or scientific purposes is understandable. Another factor which may contribute to perpetuation of this emphasis on specialized areas is the lessened likelihood that the status quo in clinical areas will be disturbed by the encroachment of computers on traditional doctor/patient/nurse relationships.

A study made by McKinsey and Company[3] on the profitability of computers employed by large companies has clear relevance to hospitals attempting to secure maximum returns on their investment in an automation project. The study found that many large companies have unprofitable computer installations because "technicians", not "managers", control the ways they are being used. The report further states that unless companies go beyond the "super clerk" uses of computers and apply the machines to crucial management and operations problems, the heavy expense of a computer installation will probably not be justifiable.

From the findings of the McKinsey study emerge three fundamental principles pertinent to the problems and organization of a hospital:

1. The direction taken toward design of a computer project should follow what engineers recognize as the "Systems Approach".[4] Simply stated, this calls for consideration of total systems needs before proceeding to the selection and design of compatible components.
2. Development of a hospital management information system calls for the commitment of the highest level medical and administrative staff to the direction and support of the project, thus securing a perspective which visualizes the relationship of the many functions, or sub-systems, of the hospital to each other and to the total needs of the hospital.
3. Interdisciplinary specialists should be members of the development team; i.e., physicians who understand the problems of data processing; researchers who understand the problems of the physician and the programmer; accountants who see the relationship between other hospital sub-systems and their own; systems specialists who can relate successfully to the other members of the group.

The Institute of Living patient centered information system has departed from the standard pattern seen in the majority of hospital systems which center around and cater primarily to a single component of the hospital complex or sets of isolated components. Adherence to the "Systems Approach" is exemplified in its stated objectives:

1. To provide a more effective method than otherwise possible through traditional means, to record, communicate, and display, when required, a comprehensive and dynamic profile of patient progress. This is accomplished through provision of simple inquiry procedures with which the user can extract an almost limitless variety of information through an on-line terminal.
2. To streamline administrative procedures, thereby freeing professional personnel for more productive purposes in the patient care and hospital management areas.
3. To increase the efficiency, economy, and safety of patient logistics; i.e., scheduling of medications, meals, diagnostic procedures, patient privileges, etc.
4. To provide a better and more sophisticated means to record, analyze, and present psychophysiological data on all types of patients for clinical and research purposes.
5. To satisfy the need for financial, personnel, and other conventional business applications; i.e., payroll, accounts receivable, personnel data, financial reports, accounting, inventory control, etc.

Progress toward these objectives has been achieved in the implementation of a hospital dependent, reactive system which has been in daily operation for two years serving the clinical, administrative, and research needs. The system includes the following features:

1. An automated master patient record file of current and historical information on all in-hospital patients (see Figure 1).
2. Twelve key areas of the hospital are equipped with Cathode Ray Tube terminals with alpha-numeric keyboards which hospital staff members use to update the patient file and request information as needed (see Figure 2).
3. A privacy feature has been incorporated to insure positive identification of a person attempting to operate a terminal and to prevent unauthorized personnel from entering or receiving privileged information.
4. The administrator, clinician, or researcher can request a report on any group of the hospital population according to desired parameters receiving an immediate reply; for example, sex, age, diagnosis, religion, origin, and/or length of stay. An example is shown in Figure 3.

```
DEMOGRAPHIC INFORMATION:  Patient's residence,
     marital status, religion, education,
     occupation, birthdate, sex, citizenship,
     social security number.

ADMISSION DATA:  Date, legal basis, type of
     admission, time and place of previous
     patient stays, referring party, personal
     physician (office and residence).

LOGISTICS:  Residence, group, observation,
     and doctor assignments (current and
     history), date and type of privilege,
     place, time, and frequency of departures,
     visits, and returns.

QUANTITATIVE BEHAVIORAL ANALYSIS:  MMPI, MHPA,
     scores and rater ID, (current and history),
     nursing note scores, special behavior
     data.

MEDICAL EVALUATION:  Duration of illness,
     drug study group code, diagnosis, medi-
     cation data, mental status.

BILLING:  Room and medication rates, charges
     and credits, current balance, billing
     party for all or specified charges.

FAMILY AND FRIENDS:  Responsible party, emer-
     gency notification, other relatives
     (address, occupation, and telephone
     number of above persons).

PHYSICAL MEDICAL DATA:  Neurologic and physical
     exam results, blood type, drug sensitiv-
     ities, other abnormalities.
```

Figure 1—Master patient record file data

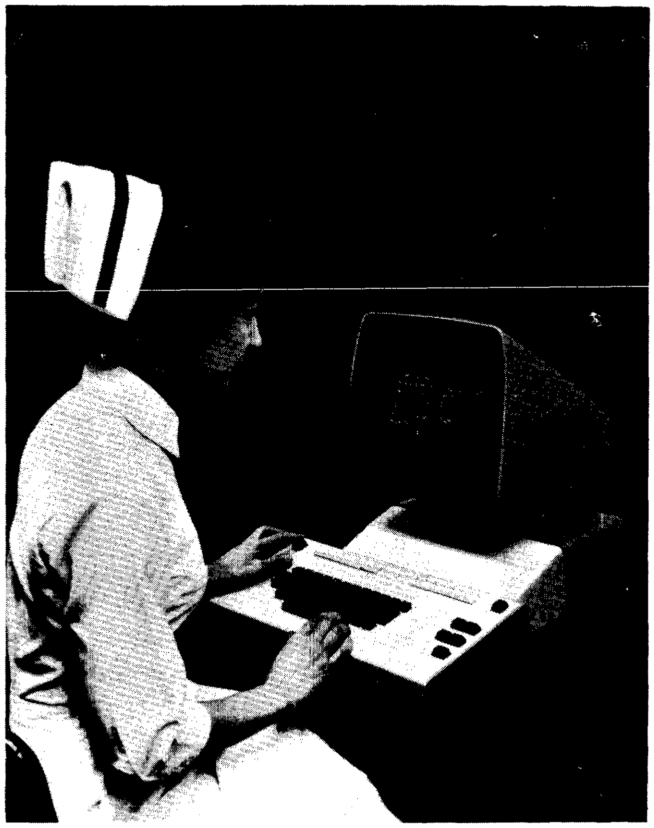


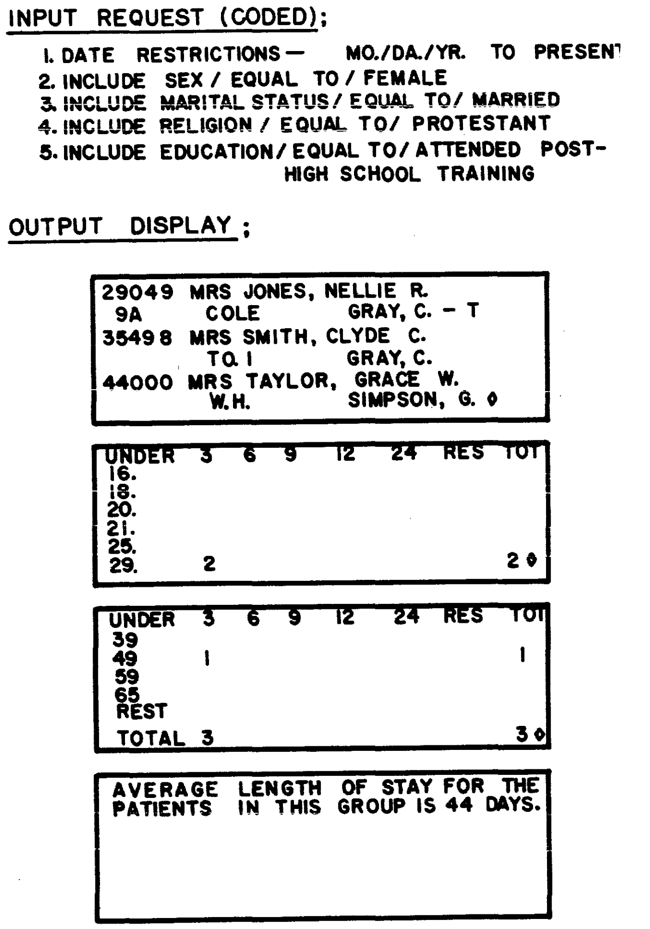
Figure 2—On-line cathode ray terminal

5. Since daily behavioral observations are the basic laboratory data for psychiatric patients, a patient's behavioral status, over time, can be graphically displayed as desired using factors derived from an automated nursing note system, as shown in Figure 4. These "fever chart" displays have been found especially meaningful and useful to clinical personnel.[5] Figure 5 is another example of a similar display comparing two patients on the factors derived from the scores achieved on the Minnesota-Hartford Personality Assay.
6. Computer aided instruction is provided through interactive procedural aids displayed on the terminal screens. Examples are the availability of a glossary to define terminology presented in an on-line mental status description and a self-administered test in the use of the newly adopted international list of diagnostic categories.
7. Automation of pharmacy orders provides a record of patient medications, produces labels for the pharmacist as a by-product, and notifies clinicians when it is necessary to renew a drug order.
8. Payroll, patient billings, census, and other related applications are also provided.

The programs, written in Autocoder, run on an IBM 1440 computer utilizing 16,000 characters of core storage, three 1311 disk drives, a 1442 card reader/punch, and a 1443 printer. A Bunker-Ramo 200 Display System provides the on-line capability with Bunker-Ramo 204 Display Stations and a Teletype Corporation Model 35 RO. By the end of the year it is anticipated the system will be transferred to a Univac 494 with central processor backup, Fastran drums and tape storage. The number of terminals will be expanded to approximately 40 in order to provide complete coverage of all hospital areas.

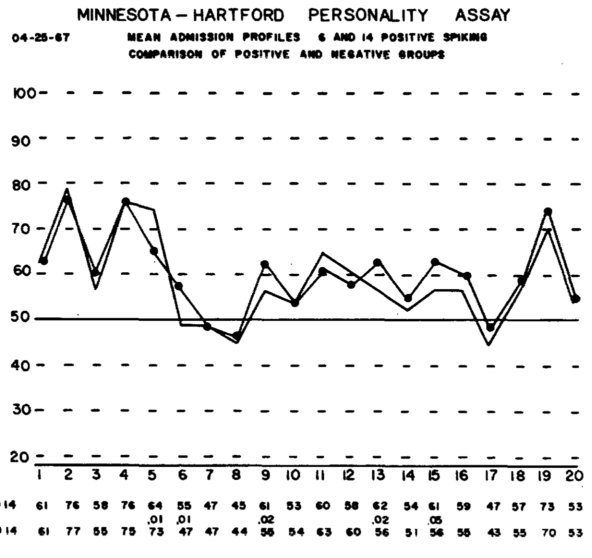
In retrospect, the following three premises on which system design philosophy has been based appear as crucial factors.[6,7,8,9]

First, the patient, his care and treatment, have been recognized as the center of the hospital universe. Every activity is directed toward this end; no machine or automated procedure is permitted to disturb the therapeutic relationship between the patient and his environment.



```
INPUT REQUEST (CODED);

  1. DATE   RESTRICTIONS —     MO./DA./YR.   TO  PRESEN
  2. INCLUDE  SEX / EQUAL  TO / FEMALE
  3. INCLUDE  MARITAL STATUS / EQUAL  TO/ MARRIED
  4. INCLUDE  RELIGION / EQUAL  TO/ PROTESTANT
  5. INCLUDE  EDUCATION/ EQUAL TO/ ATTENDED  POST-
                          HIGH SCHOOL  TRAINING

OUTPUT  DISPLAY ;

  29049  MRS JONES, NELLIE R.
   9A       COLE          GRAY, C. - T
  35498  MRS SMITH, CLYDE  C.
            TO. I         GRAY, C.
  44000  MRS TAYLOR,  GRACE  W.
            W.H.          SIMPSON, G. 0

  UNDER   3    6   9    12    24    RES   TOT
  16.
  18.
  20.
  21.
  25.
  29.     2                              2 0

  UNDER   3    6   9    12    24    RES   TOT
  39
  49      1                              1
  59
  65
  REST
  TOTAL   3                              3 0

  AVERAGE  LENGTH  OF  STAY FOR  THE
  PATIENTS  IN  THIS  GROUP IS 44 DAYS.
```

Figure 3—An example of the master patient file "extractor" capabilities



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6 AND 14 YES	61	76	58	76	64	55	47	45	61	53	60	58	62	54	61	59	47	57	73	53
					.01	.01			.02				.02		.05					
6 AND 14 NO	61	77	55	75	73	47	47	44	56	54	63	60	56	51	56	56	43	55	70	53

Figure 5—A patient behavior profile produced by scoring twenty factors of the Minnesota Hartford Personality Assay. The normal profile is equal to a score of 50 on each factor with a standard deviation of 10

Accordingly, from the point of view of both the systems engineer and the hospital, a patient centered information system provides the most logical approach to a viable system, as illustrated by Figure 6.

Secondly, the Human Factor is a critical element in the success of the computer project. The degree of

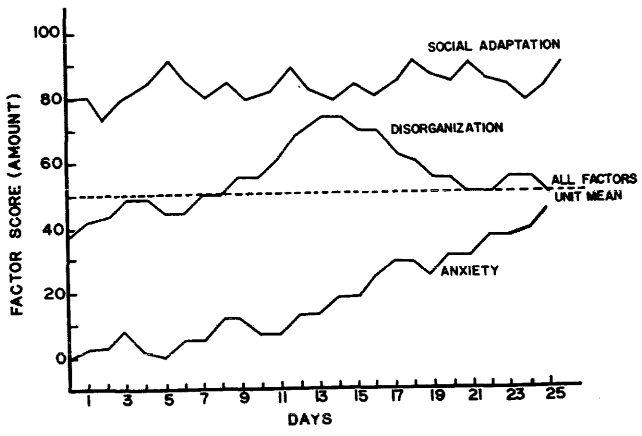


Figure 4—A longitudinal 25-day display of three nursing note behavioral factors for a sociopathic patient compared with the group mean (=50). Compared with the group mean, this patient was more socially adaptive and showed an increase in anxiety over the 25-day period. The psychiatrist believed that therapy would not be effective until anxiety had been mobilized

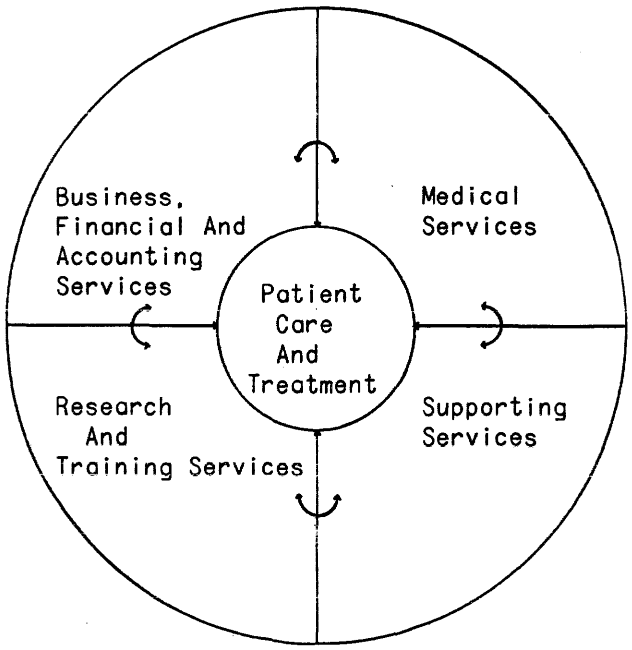

Business, Financial And Accounting Services
Medical Services
Patient Care And Treatment
Research And Training Services
Supporting Services

Figure 6—A schematic illustration of the patient-centered-hospital system concept

appropriateness and acceptability of new techniques is directly proportioned to the understanding and contributions of the users in their development.[10]

To this end, all levels of hospital staff, from top medical and administrative down, who might in any way come into contact with the system or be affected by it, have been involved in the design effort. This involvement has included on a continuing basis the following, as appropriate to the needs of the individual:

1. Exposure to the potential usefulness of computers in the care and treatment of patients through booklets, visual aids, lectures, group discussions, visits to other hospitals, etc.
2. Solicitation of ideas, even if apparently far-fetched, as to how a computer might be used in the hospital.
3 Assistance and advice in setting objectives and priorities for their accomplishments.
4. Active participation in the development of automated procedures by those involved; for example, pharmacists, nurses, the chairman of the pharmacy committee, and data processing personnel were closely involved in design and implementation of the automated medication file.
5. Frequent personal consultation by project staff members with the users of the system for their guidance and advice especially to ensure the proper use of the unique terminology employed by clinicians, nurses, and other professionals.

It has been observed that the mere involvement of key personnel from all departments can of itself improve hospital efficiency and patient care procedures, automated system or not.

Thirdly, the man/machine interface plays a vital role in the successful integration of an automated system into the hospital environment.

It is recognized that the users of the system need to enter, receive, and utilize information comfortably, effectively, and preferably without an intermediate filter, such as a clerk, ward secretary, nurse, or other personnel. For the majority of the hospital staff their only contact with the computer will be the means of communication. Quite literally, to them this is the computer.

In a more conventional business environment, a process control situation, or a computational research effort, such emphasis on choice of communications device may not be warranted. In these areas one can expect clerks, engineers, or scientists to be familiar with various types of terminal equipment. Where one must deal with personnel both unfamiliar with and often resistant to "machines", a psychological, and often a physical, barrier is interposed to the free flow of information on which the system depends.

By centering the system on the patient rather than on the "computer", new and productive patterns of staff interaction have been encouraged, leading to a re-analysis of every dimension of hospital operation that affects patient care. We conclude that the ultimate potential for applying the patient centered concept depends not on hardware which is already planned or available, but, most importantly, on capable medical, paramedical, research, and data processing personnel who can use the computer as a means and not an end for crossing traditional interdisciplinary barriers in the care of patients.

This does not mean the potential of the computer to monitor, condense, and analyze information is being relegated to a secondary level of consideration. With the acquisition of adequate third generation hardware we plan to use the data base now being accumulated for continuous monitoring by the computer for essential missing data, "essential" being determined by continuous comparison of already received data with paradigms of illness or treatment prediction patterns. We assume the essential information will vary considerably from one clinical problem to another, and hope to exploit the resources of advanced computer technology to make the models as varied and numerous as necessary.

We would judge that the ultimate success of an information system can be evaluated by the degree of increased communication and interaction it encourages amongst these various specialists; this criterion implies that *all* relevant judgments would be used in the care of the patient. Early evaluation of such a system at the Institute of Living demonstrated a significant increase in such communication.[11] This approach has used the computer as something more than an instrument for automation.

ACKNOWLEDGMENTS

The computer project at the Institute of Living has been supported in part by NIMH Grant MH-08870, Bernard C. Glueck, Jr., M.D., Principal Investigator and Director of Research. This research was also supported by NIMH Grant MH-08557 and by the Gengras Foundation.

The authors acknowledge the assistance of Mr. R. Peter Ericson, Miss Dorthie McIntyre, and Mrs. Dorothy E. Reiss.

REFERENCES

1 C F STROEBEL W L BENNETT R P ERICSON B C GLUECK JR
Designing psychiatric computer information systems: problems and strategy
Comp Psychiat Vol 8 No 6 1967

2 R H GIESLER
How many hospitals use automatic data processing equipment
Hospitals JAHA Vol 38 January 1964

3 Mc KINSEY AND COMPANY, INC
Unlocking the computer's profit potential

4 W L BENNETT
The systems approach to hospital automation
Hospital Financial Management J HFMA January 1969

5 M ROSENBERG B C GLUECK JR W L BENNETT
Automation of behavioral observations on hospitalized psychiatric patients
Amer J Psychiat 123 1967 926–929

6 B C GLUECK JR C F STROEBEL
The computer and the clinical desicion process: II
Supplement to the American Journal of Psychiatry Vol 125 No 7 January 1969

7 W L BENNETT
How to live with a computer and like it
Medical Record News JAAMRL Vol 39 No. 3 1968 42–44

8 W L BENNETT J A HOUCK
A three step plan for automation
Hospitals JAHA Vol 41 1967 61–66

9 W L BENNETT
A viable computer-based hospital information system
Hospital Management Vol 103 1967 43–47

10 M ROSENBERG R P ERICSON
The clinician and the computer-affair, marriage, or divorce?
Supplement to the American Journal of Psychiatry Vol 125 No 7 January 1969

11 M REZNIKOFF D HOLLAND C F STROEBEL
Attitudes toward computers among employees of a psychiatric hospital
Mental Hygiene Vol 51 1967 419–425

A position paper—Computers in medicine: Automation vs. improvement of status quo

by ALVAN R. FEINSTEIN

Yale University School of Medicine
New Haven, Connecticut

Computers have thus far been applied in medicine most effectively in situations where a standard mechanism already exists for dealing with the data: in the accounting problems of administrative work; in sorting and printing out the results of laboratory tests; and in conventional types of mathematical calculation performed during research or other activities. Despite these obvious and desirable successes, computers have not yet had an important impact on the more inherently clinical features of medical strategy and tactics. The intellectual qualities of scientific practice in clinical medicine do not appear to have been significantly affected by the many theoretical models and grandiose systems proposed during the recent exuberance of "computers in medicine."

Perhaps the greatest barrier to true progress in applying computers to clinical problems is the premise that satisfactory scientific approaches now exist for acquiring medical data and for interpreting the data during diagnostic and therapeutic decisions. Since this premise is not valid, enormous amounts of time, effort and money may be expended in constructing computerized systems and models that will be obsolete or inadequate for the real needs of clinical medicine.

Computational approaches to diagnosis might have been useful 40 years ago, but are often obviated today by the modern precision of individual pathognomonic paraclinical tests. With the aid of radiography, biopsy, cytology, endoscopy, and diverse laboratory procedures, many diseases can be accurately identified today without the use of complex inferential logic or statistical computations of probability. The major intellectual need in diagnosis today is for a better way of choosing the tests rather than calculating the names of diseases. Such improved strategies cannot be devised and checked, however, until appropriate clinical algorithms are constructed to demonstrate the flow of logic in the diagnostic "work-up," and until appropriate medical data are assembled to show the values and risks of paraclinical tests at each step in the logical sequence. The construction of such algorithms and the assemblage of such data require attention not to computers and statistics, but to the basic ingredients of clinical reasoning and activities.

Current systems for acquiring and storing the data of patients' histories are ingenious but inadequate because the types of data, the necessary descriptive constituents, and their subsequent tactics of application are not clearly recognized or defined. Many critical types of information—such as iatrotropic stimuli, subtle nuances in symptom descriptions, sequential patterns of symptoms, and the effects of co-morbid ailments, as well as the entire class of communications that are transmitted non-verbally—are omitted from information now being stored in automated systems, and inadequate attention has been given to the different rational procedures that must be used when the same data are analyzed for different clinical purposes.

The automated interpretation of electrocardiograms and of other paraclinical tests cannot be effective until specific, rigorous criteria are established and standardized for the diagnostic interpretations. Such criteria have not been developed for most of the "diseases" of modern medicine. Most of the existing diagnostic criteria for disease are derived from observations made at necropsy, but the histopathologic concepts have not been subjected to careful studies of observer variability, and precise criteria have not been formulated and accepted for integrating the combination of clinical, technologic, and morphologic data that must be used for diagnoses made in living patients.

Concepts of "normality" are currently in a state of confusion because the "normality" defined by a statistical Gaussian curve is quite different concept from the "normality" defined by a state of health or disease. Moreover, regardless of which definition is used for "normality," a satisfactory range of epidemiologic populations has not been assembled and followed as a source of reliable data for analysis.

"Support systems" for clinical decisions will be intellectually chaotic until they recognize and separate the different types of data and reasoning used for diagnosis, prognosis, and therapy. Each type of reasoning requires different data, different logic, and different criteria. Because the taxonomic categories for data are currently undefined or incomplete; because standardized interpretive criteria do not exist; and because the available data are not satisfactory either in epidemiologic range or in temporal extensiveness—the current attention to *system* rather than to *data*, *clinical logic*, and *criteria* seems directed at peripheral rather than basic issues.

By providing a magnificent means of storing, retrieving, and counting complex data, computers offer clinicians an opportunity for major improvements in the scientific practice of medicine. The opportunity will be lost if the computer is used merely to automate a defective *status quo*. A more exciting challenge of the computer is the incentive it offers clinicians to explore the basic data and complex reasoning of clinical medicine, so that intellectual clarity and precision can be established for constructing suitable clinical algorithms, for developing appropriate criteria for decisions, and for performing new research projects that will yield respectable data with which to enlighten the future rather than embellish the past.

An analytic model of multiprogrammed computing

by ROBERT R. FENICHEL

Massachusetts Institute of Technology
Cambridge, Massachusetts

and

ADRIAN J. GROSSMAN

Borden, Inc.
New York, New York

INTRODUCTION

The work described in this paper was undertaken in order to provide estimates of the throughput capacity of certain multiprogrammed computer systems. In particular, we consider those systems whose users are never simultaneously receiving input-output and central processor attention. Systems making use of "demand paging" tend to fall within this class.

Our definition of "throughput" should be made more explicit in this introduction. We initially considered a model in which the measure of system performance was response time. But this model required us to phrase each result in terms of user reaction time; the same system can give snappy service to lethargic users and inferior service to a more energetic clientele.

The current model looks to system perfomance in terms of *completed interactions per unit time.** This measure has the merit that it is indifferent to the energy and even to the total number of competing users of the system. Our only assumption in using this measure is that the number of users is sufficient to maintain a non-zero load on the system at all times.

We wish to understand the effects of each of our analytic assumptions. In particular, we make the following claim: *Each of our assumptions is optimistic.* To the extent that our model over-simplifies, it over-estimates expected system performance.

For example, we assume of many parameters that they are taken from distributions of zero variance. This is always an optimistic assumption. In every case, these parameters may be regarded as demands upon system components. The variances which we bar could only lead to the idleness of a component; utilization, like time, cannot be recaptured.

System: CPU, core, drum

We shall first consider a system whose resources are a single central processing unit (CPU), some fixed quantity of executable memory (core), and a drum.

Our first assumption is that there is a fixed, determinable number k which is the number of user programs simultaneously in core.** In any paged environment, of

* This measure is identical to that used by Gaver (*Journal of the Association for Computing Machinery,* Vol. 14, No. 3, July 1967, pp. 423–438), and it is useful to compare our work to his. Gaver's work is characterized by sophisticated statistical analysis, and Gaver's analytic assumptions were made in order to facilitate his manipulations. For example, Gaver assumes that the service times of input-output peripherals approximate an exponential distribution (page 428). While this approximation can be made to fit real system behavior quite closely, the effect of the assumption upon the whole analysis is not clear. Insofar as real equipment behaves less smoothly than the distribution suggests, is the system degraded or improved?

Many of Gaver's idealized systems differ in what prove to be only minor ways. For example, he compares systems which differ only in the variances of the distributions of their respective parameters. It is greatly to Gaver's credit that these comparisons were published, but we perceive them as strictly negative results. That is, Gaver has shown that different distributions really are not ever very different.

** To take account of shared code, we prorate its ownership among its users in the obvious way.

course, programs may be only partially in core. We count a given program among the k if, and only if, it is either (1) now using the CPU, (2) waiting for the CPU and sufficiently loaded so that the system would transfer the CPU to it were the CPU free, or (3) not using or waiting for the CPU (that is, using or waiting for some other component of the system) and as fully loaded into core as the least-loaded program in condition (1) or (2).

Certainly, k is bounded by the physical size of core; each core-resident user needs at least the space which will be necessary for storage of his current machine conditions. But in order to decrease the frequency of page requests, the system's supervisor will probably limit k so as to increase the quantity of core provided to each core-resident user. If the supervisor wisely attempts to allot core space for the working-set[1] of each user, k may have to be quite small indeed.[2]

During the time that a user's program is in core, its state will cycle between

α, waiting for or receiving CPU service, and
β, waiting for or receiving a single page from the drum.

We specifically ignore, in this analysis, all core requirements of other peripheral devices which may be present. For example, we assume that if one of the users comes to wait for typewriter action, he effectively disappears from the system. In particular, such a user is not tying up any of the k blocks of valuable core.

We assume that during each interaction, each user will require exactly C seconds of CPU attention. Similarly, we assume that each user will require that D pages be fetched from the drum on his behalf during every interaction. We specifically ignore the problem of *writing* pages back onto the drum.† We assume, moreover, that in a single-interaction cycle of

$$\alpha_1 \, \beta_1 \, \alpha_2 \ldots \alpha_D \, \beta_D \, \alpha_{D+1}$$

the C seconds of CPU attention are divided among the intervals of state α into $D + 1$ *equal* pieces.

Our fundamental concern is with the rate at which user programs leave state α; that is, the rate at which users cease to require CPU service.

We know, after all, that only $D/D + 1$ of this flow is toward state β. The remainder of the flow from α consists of users who have completed their use of core.

† This assumption is not necessarily very generous. Especially in systems utilizing much pure-procedure programming, the read/write ratio may be very high. Also, reading is inherently more difficult than writing, since an item to be read must be read from where it is written, whereas an item to be written can well be placed where convenient.[3]

This flow out of core is exactly the rate at which the system produces evidence of useful work.

In order to compute these flow-rates, we shall need to compute the occupancy levels a and b, where a is the number of user programs in state α and $b = k - a$ is the number of user programs in state β.

It is convenient to discuss the system in terms of the following figure:

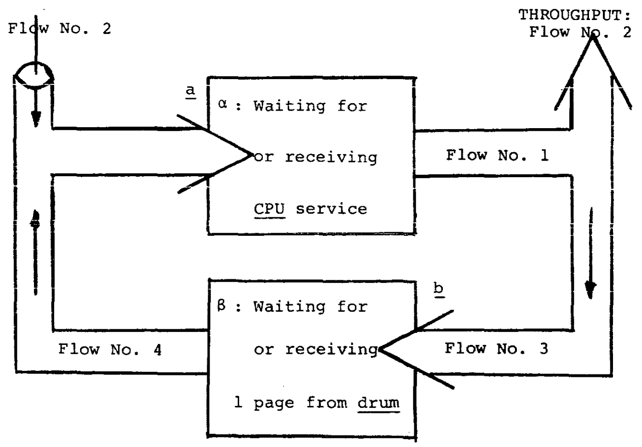

Flow No. 2
THROUGHPUT: Flow No. 2
a
α : Waiting for or receiving CPU service
Flow No. 1
b
β : Waiting for or receiving 1 page from drum
Flow No. 4
Flow No. 3

Figure 1—Flows: CPU/drum system

In terms of this diagram, we have already stated that

1. $\text{Flow No. 2} = \dfrac{\text{Flow No. 1}}{D + 1}$

2. $\text{Flow No. 3} = \dfrac{D \times \text{Flow No. 1}}{D + 1}$

3. $a = k - b$

Moreover, over any long run it must be true that the number of user programs which have arrived at state β is exactly equal to the number of user programs which have left that state. The most optimistic possible assumption (by the old congestion/idleness argument) is that the rates of flow Nos. 3 and 4 are constant over time. But then,

4. $\text{Flow No. 3} = \text{Flow No. 4}$

Now to solve for the unknowns of the diagram (that is, for a, b and the four rates of flow) it will be necessary to determine the functions f and g such that

5. $\text{Flow No. 1} = f(a)$

6. Flow No. 4 $= g\ (b)$

We can begin to describe f right away. If a is zero, naturally f is zero too (that is, the CPU disposes of zero users per second when there are zero users who want to use it). When a is an integer greater than zero, the CPU disposes of a user every $C/D + 1$ seconds; the flow is just $D + 1/C$ users per second.

But what if a is non-integral? For example, suppose that only one user program can fit into core. If D is not zero, then the average number of programs seeking CPU attention must surely be properly between one and zero. The most optimistic interpretation we can make is that $f\ (a)$, for a non-integral, may be computed by linear interpolation from the values of f at the surrounding integral values of a.

Thus, in summary, we have as the CPU output rate'

7a. $f\,(a) = 0$ [if $a = 0$]

7b. $f\,(a) = (D + 1)/C$ [if $a \geq 1$]

7c. $f\,(a) = a \times (D + 1)/C$ [if $0 < a < 1$]

The function $g\ (b)$ is almost as easy to develop. For any integer b_o, it is a trivial simulation task to find the output of a given drum under the assumption that exactly b_o entries are always in its queue. The points obtained rise steeply at first, but then they level out to an asymptote at the capacity of the drum. Because the second differences of these points are all negative, it is once again optimistic to fill in the curve by means of linear interpolation between adjacent points. That is,

8a. $g\ (b) = 0$ [if $b = 0$

8b. $g\ (b)$ is determined by drum simulation [if b is integral]

8c. $g\ (b) = g\ (\text{entier}\ (b)) + (b - \text{entier}\ (b)) \times (g(\text{entier}\ (b) + 1) - g\ (\text{entier}\ (b)))$ [if b is not integral]

Now that f and g are known, a and b are uniquely determined by equations (1) through (6); standard approximation methods can be used to solve:

9. $$\frac{Df(a)}{D+1} = g\ (k - a)$$

One approximation method is displayed in Appendix A.[4]

System: CPU, core, drum, disk

The system could even be more complicated, and the same technique would work. For example, a new f function could easily take account of any number, n, of anonymous processors. We could even add other peripheral devices; the CPU-drum system might stand to a disk, say, as the CPU alone stands to the drum. Such a system would look like:

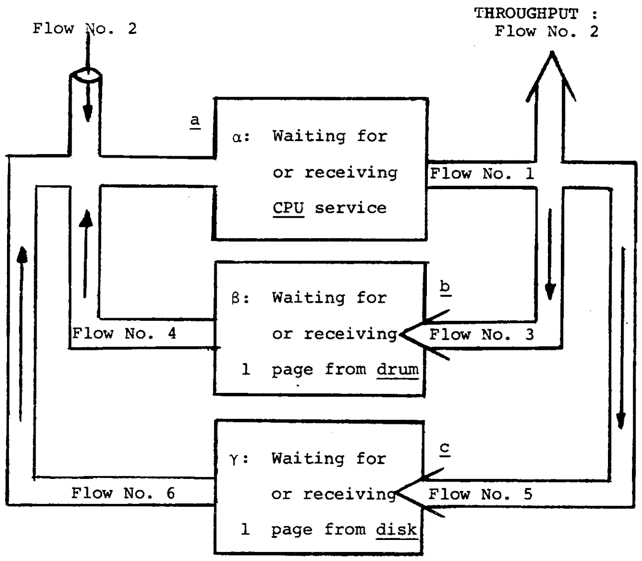

Flow No. 2
THROUGHPUT : Flow No. 2
a
α: Waiting for or receiving CPU service
Flow No. 1
b
β: Waiting for or receiving 1 page from drum
Flow No. 4
Flow No. 3
c
γ: Waiting for or receiving 1 page from disk
Flow No. 6
Flow No. 5

Figure 2—Flows: CPU/drum /disk system

With $a + b + c = m$ user programs in core, Ds disk pages read per interaction, and

10. Flow No. 6 $= h\ (c)$

for some function h like f and g, we can determine h by simulation of the disk and then we can compute a, b and c as we computed a and b above (see Appendix B[5]).

When a, b and c have been calculated, a great deal of information is available. Since the best possible flow rates from system components are simple functions of the physical characteristics of the component, utilization levels can be computed. Thus,

$$\text{CPU utilization} = \min\left(1, \frac{a}{n}\right) = \frac{f\ (a)}{(D + Ds + 1) \times n}$$

$$\text{Drum utilization} = \frac{g\ (b)}{\text{drum rotational speed in pages/second}}$$

$$\text{Disk utilization} = \frac{h(c)}{\text{best possible disk transfer rate in pages/second}}$$

One can even compute the core utilization of the modeled system, according to the following argument:

> All the core in the system (m) is distributed between rotating storage (b, c) and the CPUs (a). No core in the former class can be considered idle; every addition to the drum queue definitiely makes the drum more effective, and similarly for the disk. If any of this core were removed, the system would suffer.

Of the a user programs grouped around the CPUs' however, only n are receiving service. The excess users (if $a > n$) are just waiting, and their presence does not improve the effectiveness of the CPUs. If this excess core is removed, the model shows no change in throughput. That is,

$$\text{Core utilization} = \min\left(1, \frac{n+b+c}{m}\right)$$

Utility of the model

We believe that this model has three distinct sources of utility.

1. Our assumptions of zero-variance behavior may be reasonable for some special-purpose systems (e.g., reservation systems). The model should accurately predict the behavior of such systems.
2. However close the model's assumptions are to the realities of other systems, these other systems can surely do no better than the model expects them to. The model provides upper-bound values for the performance of these systems.
3. The model suggests at least qualitative explanations for many complex system phenomena. For example, the model allows a simple explanation of how increasing paging demands might be met by increasing CPU performance.

ACKNOWLEDGMENTS

The work described in this paper was initiated while the authors were consultants to the MEDINET Department of the General Electric Company. Several discussions with MEDINET employees, particularly T. N. Hastings, were seminal to the work. P. J. Denning of M.I.T. reviewed an early draft of the paper and made many helpful suggestions.

The work was supported by Project MAC, a Massachusetts Institute of Technology Research Program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract No. NONR-4102 (01).

REFERENCES

1 P J DENNING
The working set model for program behavior
Communications of the Association for Computing Machinery Vol 11 No 15 May 1968 323–333

2 L C VARIAN E G COFFMAN
Further experimental data on behavior of programs in a paging environment
Communications of the Association for Computing Machinery Vol 11 No 7 July 1968 471–474

3 P J DENNING
Effects of scheduling on file memory operations
Proc S J C C 1967

APPENDIX A

Sample ALGOL routines for computing a and b given k

```
real procedure f(a);
real a; value a;
begin if a = 0
      then f: = 0
    else if a ≥
      then f: = (D + 1)/C
    else f: = a × (D + 1)/C;
end;
real procedure g(b);
real b; value b;
begin own real array ig [0:k];
comment The array ig is magically filled by simulation.
        The entry ig [n] is the proper value of g(b) for
        b = the integer n;
real eb;
    eb: = entier (b);
    if b = eb
      then g: = ig [b]
    else g: = ig [eb] + (b-eb) × (ig [eb + 1] − ig [eb])
end;
procedure Distribute k (k, D, C, a, b, tolerance);
comment Half-interval attack upon k = a + b;
real k, D, C, a, b, tolerance;
value k, D, C, tolerance;
begin real frac, CPU output, drum input, drum output;
  a: = k;
  frac: = k/2;
aset: b: = k − a;
  CPU output: = f(a);
  drum input: = D × CPU output/(D + 1);
  drum output: = g(b);
  if drum input > (1 + tolerance) × drum output
    then a: = a − frac
    else if drum input < (1 − tolerance) × drum output
      then a: = a + frac
    else go to done;
  frac: = frac/2;
  go to aset;
done: end Distribute k;
```

APPENDIX B

Sample ALGOL routines for computing a, b and c given m

```
procedure Distribute m (m, C, D, Ds, tolerance, n, a,
          b, c, CPU output, drum output, disk output,
          throughput);
real m, C, D, Ds, tolerance, n, a, b, c, CPU output,
    drum output, disk output, throughput;
value m, C, D, Ds, tolerance, n;
begin real k, frac, disk input;
  procedure Distribute k;
  begin real frac, drum input;
    real procedure f;
    begin if a ≥ n
        then f: = n × (D + Ds + 1)/C
      else f: = a × (D + Ds + 1)/C
    end f;
    b: = k;
    frac: = k/2;
  seta: a: = k − b;
    CPU output: = f;
    drum input: = (D/(D + Ds + 1)) × CPU output;
    drum output: = g(b);
    if drum output > (1 + tolerance) × drum input
        then b: = b − frac
      else if drum output < (1 − tolerance) × drum input
        then b: = b + frac
      else go to donek;
    frac: = frac/2;
    go to seta;
  donek: end;
  c: = m;
  frac: = m/2;
setk: k: = m − c;
  Distribute k;
  disk input: = (Ds/(D + Ds + 1)) × CPU output;
  disk output: = h(c);
  if disk output > (1 + tolerance) × disk input
      then c: = c − frac
    else if disk output < (1 − tolerance) × disk input
      then c: = c + frac
    else go to donem;
  frac: = frac/2;
  go to setk;
donem: throughput: = CPU output/(D + Ds + 1);
  end
```

Measurement based automatic analysis of FORTRAN programs *

E. C. RUSSELL and G. ESTRIN

University of California
Los Angeles, California

INTRODUCTION

A graph model of computer programs has been developed in a series of studies[1–5] directed toward improving analysis of the structure of programs executed on different computer configurations. One inherent weakness of the model has been the need for estimates of the mean number of times a program would cycle around its loop structures and estimates of branching probabilities. Extensive improvements were made in the model on the assumption that good estimates would be inserted during a manual transformation of a given program into a computer processable graph representation. The combination of improved tools for measurement of program activities and recently developed analysis programs now permit automatic analysis of source programs. The automatic analysis is based on more reliable measured a priori statistics. This paper discusses a valuable by-product of this measurement and analysis which directs attention toward those parts of a program which are leading candidates for application of optimization techniques. In particular we present an example of the automatic analysis of programs written in the FØRTRAN IV language. FØRTRAN was selected as a first target for analysis because there exists a large number of time-consuming programs written entirely in FØRTRAN.

The first part of the paper presents a simple summary of the graph model of programs. The second part deals with a particularization of this model to FØRTRAN programs. The last part presents the results of an experiment which illustrates the use of the graph model based on measurements. Automatic analysis procedures obtain an ordering of FØRTRAN statements according to their frequency and time-of-execution along with other structural data.

The computational model

A principal goal of the several investigators[1–5] who have developed a graph model for representing computer programs has been to study the structure of the program for analysis of its space and time requirements on a variety of hardware configurations. This model is illustrated in Figure 1.

The salient properties of the model are:

1. There is one original vertex (v_1)
2. There is one terminal vertex (v_3)

(When either of these conditions does not exist, a pseudo-vertex is appended. Thus multiple terminal vertices are connected to a pseudo terminal vertex and multiple origin vertices are preceded by a pseudo-origin vertex.)

3. Each vertex is connected to other vertices by directed arcs. There may be more than one arc directed either to or from a vertex, in which case the interaction of the arcs is specified by a logic condition. Arcs may enter a vertex or leave a vertex in a simultaneous ("AND") fashion or in an exclusive ("OR") fashion. The "AND" condition indicates situations in which multiple-processors may be used to advantage.
4. Cycles are represented by arcs which are directed from v_j to v_i where $j \leq i$. This implies that care is taken in the choice of numbers for vertices.

Thus far only structural properties of the program are described. The remainder of the description is concerned with the definition of the operation to be performed at each vertex. Such a description includes:

* This research was sponsored by the Atomic Energy Commission [AT(11-1) Gen 10 Project 14], the Advanced Research Projects Agency ARPA SD 184, and the Office of Naval Research, Information Systems Division. Reproduction in whole or part is permitted for any purpose of the United States Government.

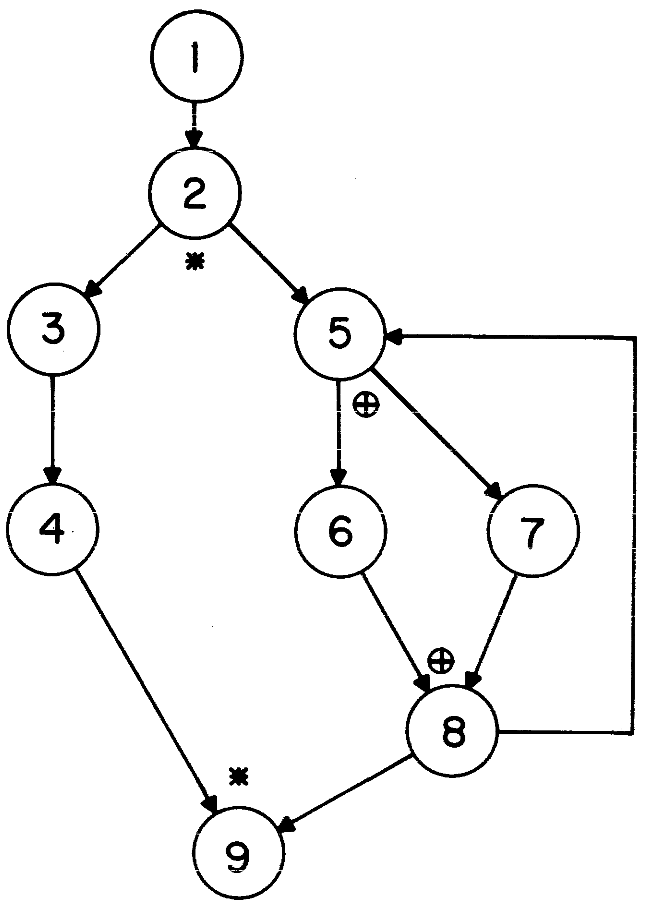


Figure 1—Graph model of a computation

5. The block of instructions required to perform the computation.
6. The identification of required input data.
7. The identification of required output data.
8. The amount of intermediate (temporary) storage required.

One of the goals of this modelling process is the a priori analysis of the execution of a program on a particular hardware configuration. In order to complete the program description for this purpose, the following additional parameters are required:

9. For each "exclusive or" output, a probability for each arc.
10. The iteration factor for each cycle (These two parameters may be derived from a single "frequency count" for each vertex.)

This model has the advantage over some others (Rodriguez[8]), that only a single type of vertex exists. Variations such as input connections, output connections, actual computations, etc., are described parametrically. This leads quite naturally to the description of the graph in terms of Boolean matrices.

Additional generalizations are also possible. In the experiments performed, FORTRAN statements are taken as the primitive computations represented as vertices. It may not always be desirable to make such a choice. In highly parallel computations, a refinement or "blowup" of a particular vertex may reveal parallel processing potential within complex statements. Conversely, large portions of the program may represent such a minor portion of the computation time, that it may be helpful to collapse portions of the graph into single vertices. Such transformations are possible, given that all the above descriptive parameters are available. For the present an entire FØRTRAN program is analyzed and represented as a single graph. If subprograms are referenced, they must be described parametrically prior to their inclusion. The parametric description of a subprogram can be developed independently and a library of subprograms prepared.

Particularization to FORTRAN programs

Nomenclature

Vertex definition

The application of the model to FORTRAN will be made at the statement level. That is, each executable FORTRAN statement will be assigned to a unique vertex. Other choices could be made: for example, if more than one vertex per statement were permitted then each arithmetic operator could be assigned to a unique vertex and analysis of parallel arithmetic expressions could be performed. This would result in graphs with large numbers of vertices. On the other hand, a single vertex could be made to represent several statements or even an entire subprogram. This would provide for smaller graphs but could obscure potential parallelism. With an initial analysis at the statement level, a refined analysis could then be performed on those portions of the program which consume significant portions of execution time, whereas less significant portions can be collapsed into fewer vertices.

Each program will contain one *origin vertex* (w_1) which is predecessor to all other vertices in the graph. (A program with multiple entries will still have one pseudo-origin with immediate outbranchings to the various entry points.)

Each program will contain one *terminal vertex* (w_n) which is successor to all other vertices in the graph. (Multiple exits from the program will be tied to this one terminus.)

For each vertex, the *input data set*, I_i, consists of the names of those variables which are referenced by the statement represented by the vertex. For example, for an arithmetic assignment statement this includes all variables in the expression and any subscripts.

For each vertex, the *output data set*, O_i, consists of the names of those variables which are modified by the execution of the statement represented by the vertex.

Arc definitions

Three categories of connections between vertices will be established; sequential connections, logical connections and loops.

Each executable FØRTRAN statement which does not explicitly specify the successor statement implicitly "falls through" to the next executable statement in the program. These connections (of the form (w_i, w_{i+1})) are designated *sequential connections*. There can be only one sequential successor vertex for each vertex.

Each executable FØRTRAN statement which does specify explicitly one or more successor statements produces connections which are designated *logical connections*. The connections are of the form (w_i,w_j) where $i \neq j$.

Some of the arcs of a program represent the return path of a programmed loop. Of course, any one arc in a cycle can be selected as the return path but often one is recognizable from the syntax of the language (e.g., the FØRTRAN "DO" statement). Each arc which closes a programmed loop is designated as a *feedback connection*.

Graph representation of FORTRAN IV

A complete specification for the graph representation of FORTRAN statements is given in the Appendix.

Experimental measurements

Two attributes of the vertices of a graph are determined from experimental measurement. These are: (1) vertex activity number and (2) vertex single-execution time.

Vertex activity in a sequential program

Vertex activity can be determined by making some minor modifications to the source program before execution. These modifications are similar to those outlined in "SNUPER COMPUTER, A Computer Instrumentation Automaton," Estrin.[9] For these experiments, however, only vertex activity and loop activity are desired and not all arc activity as in the reference. Because of this, the artifact introduced is considerably reduced. The required measurements are as follows:

1. the number of times the program is entered.
2. the number of times each labeled statement is executed with special treatment of DO loops and unusual branches.
3. the number of times the auxiliary statement of a "logical if" is executed.

From these measurements, the entire set of vertex activities and loop factors may be calculated.

Introduction of artifact

The artifact introduced into FORTRAN programs consists of a series of subroutine calls. Routine EMIT (i) is introduced to perform the activity of counting the number of times vertex i is executed. For efficient execution, the subroutine EMIT is written so as to perform the emit table address calculation and then modify the calling instructions to perform the counter incrementing with "inline" code after the first encounter of that call.

The monitoring of vertex activity must be introduced at the following points in a FORTRAN program:

1. Measurement of origin vertex:

   Following the SUBROUTINE or FUNCTION statement or preceding the first executable statement of main program, insert CALL EMIT (i). In order to properly monitor entries into a program via the ENTRY statement, the CALL EMIT which follows an ENTRY must not be executed if control "falls through" from above the ENTRY statement. For example:

```
A = B
ENTRY FIRST
CALL EMIT (i)
C = D
```

would record an erroneous measure of entry to the program at FIRST. In such a case the following is a correct modification of the program:

```
A = B
GO TO XX
ENTRY FIRST
CALL EMIT (i)
XX CONTINUE
C = D
```

2. Measurement of labelled statement:
   Replace labelled (executable) statement L : s by L : CALL EMIT (n) where L is the state-

ment label (external formula number), s is the statement and n is the vertex number assigned to s.

3. Measurement of DO loops:
   Insert a CALL EMIT (n+1) after the DO statement where n is the vertex number assigned to the DO statement.
4. Measurement of unusual branches:
   In the case where CALL or READ statements provide unusual returns, the activity of the succeeding statement must be monitored. For example:

```
     CALL SUB1 (A,B,$2)
       A = D
     2 B = C
```

   The activity of the statement A = D is not known unless specifically monitored.
5. Measurement of logical IF statements:
   In order to monitor the conditional statement, it is necessary to introduce "CALL EMIT" but the restriction is a single statement as the conditionally executed branch. Therefore the following scheme is proposed (as in Estrin[9]).
   a. Negate the logical condition
   b. Replace the conditional statement by a 'GO TO' statement branching around the original conditional statement and an inserted 'CALL EMIT.' For example:

```
      IF (t) s
      becomes
      IF .NOT. (t) GO TO XXX
      CALL EMIT (i+1)
      s
      XXX CONTINUE
```

   c. On the other hand, if s cannot pass control to the next executable statement, the control is simpler:

```
      IF (t) s
      becomes
      IF (t) s
      CALL EMIT (i+2)
```

   where w_i is the vertex assigned to the IF statement.

Two problems in implementation now occur. If this logical IF is the terminal statement of a DO loop, the program meaning has been altered. Also, the statement label XXX may already have been used. If the logical IF is the terminal statement of a DO loop, the DO reference should be changed to XXX. (This might be most easily implemented by adding a unique terminal for each DO of the form XXX CONTINUE.) The uniqueness of statement labels can be guaranteed by keeping track of all statement label references or generations during the source program scan.

Subprogram inclusion

There are at least two methods of including subprograms in the analysis. The first is to do an independent analysis of the subprograms and provide only the aggregate attributes of the subprogram for inclusion in the programs which call it.

The second method is to combine the subprogram with its dynamic ascendants replacing CALL statement by GO TO XX statements where XX is the first executable statement of the subroutine (now a part of the same graph). The RETURN from a subroutine is replaced by a GO TO $(n_1, n_2, n_3 \ldots)$ IXX where n_1, $n_2 \ldots$ are the statements after each CALL to the subroutine and IXX is an index specifying which particular call is being executed. In the process of including a subprogram in the graph, all of the COMMON statements can be systematically eliminated. This involves transforming variables used in the subprogram to their calling program equivalent names and generating unique names for local subprogram variables.

Discussion of experiment

A process flow chart describing the experimental procedure is illustrated in Figure 2. The program to be analyzed goes through a process which produces two types of outputs. The first type of output (on the left) is the actual graph model of the program in machine-processable form. The second type is the modified source FORTRAN program ready for compilation and for execution during which raw frequency data is to be gathered. The post-execution program analyzer accepts the two sets of output data and produces the listed vertex and program attributes.

Figure 3 gives a listing of our example program which was written to simulate the UCLA Boolean Analyzer.[10] It has 127 vertices as defined in Part 2 of this paper. These 127 vertices are numbered on the right of the listing. Figure 4 shows the computer gen-

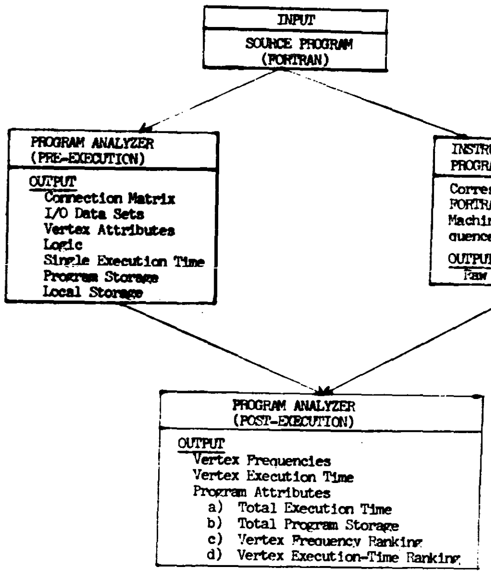
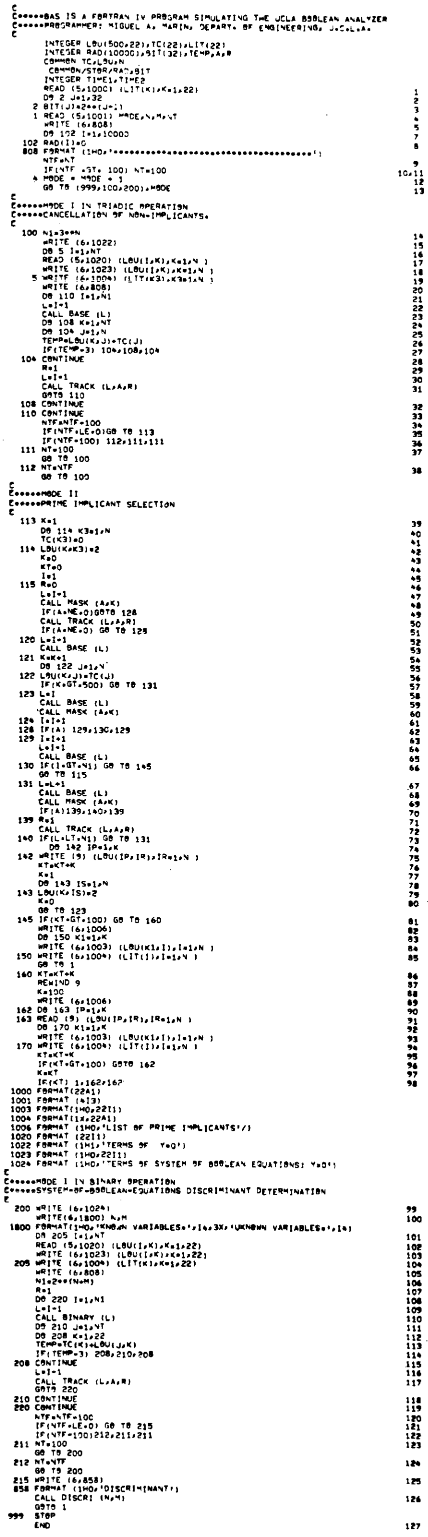


Figure 2—The measurement and analysis processor



```
C
C*****BAS IS A FORTRAN IV PROGRAM SIMULATING THE UCLA BOOLEAN ANALYZER
C*****PROGRAMMER: MIGUEL A. MARIN, DEPART. OF ENGINEERING, U.C.L.A.
C
      INTEGER LOU(500,22),TC(22),LIT(22)
      INTEGER RAD(10000),BIT(32),TEMP,A,R
      COMMON TC,LOU,N
       COMMON/STOR/RAD,BIT
      INTEGER TIME1,TIME2
      READ (5,1000) (LIT(K),K=1,22)                                  1
      DO 2 J=1,32                                                    2
    2 BIT(J)=2**(J-1)                                                3
    1 READ (5,1001) MODE,N,M,NT                                      4
      WRITE (6,808)                                                  5
      DO 102 I=1,10000                                               7
  102 RAD(I)=0                                                       8
  808 FORMAT (1H0,'****************************************')
      NTF=NT                                                         9
      IF(NTF .GT. 100) NT=100                                    10,11
    4 MODE = MODE + 1                                               12
      GO TO (999,100,200),MODE                                      13
C
C*****MODE I IN TRIADIC OPERATION
C*****CANCELLATION OF NON-IMPLICANTS.
C
  100 N1=3**N                                                       14
      WRITE (6,1022)                                                15
      DO 5 I=1,NT                                                   16
      READ (5,1020) (LOU(I,K),K=1,N )                               17
      WRITE (6,1023) (LOU(I,K),K=1,N )                              18
    5 WRITE (6,1004) (LIT(K3),K3=1,N )                              19
      WRITE (6,808)                                                 20
      DO 110 I=1,N1                                                 21
      L=I-1                                                         22
      CALL BASE (L)                                                 23
      DO 108 K=1,NT                                                 24
      DO 104 J=1,N                                                  25
      TEMP=LOU(K,J)+TC(J)                                           26
      IF(TEMP-3) 104,108,104                                        27
  104 CONTINUE                                                      28
      R=1                                                           29
      L=I-1                                                         30
      CALL TRACK (L,A,R)                                            31
      GOTO 110
  108 CONTINUE                                                      32
  110 CONTINUE                                                      33
      NTF=NTF-100                                                   34
      IF(NTF.LE.0)GO TO 113                                         35
      IF(NTF-100) 112,111,111                                       36
  111 NT=100                                                        37
      GO TO 100
  112 NT=NTF                                                        38
      GO TO 100
C
C*****MODE II
C*****PRIME IMPLICANT SELECTION
C
  113 K=1                                                           39
      DO 114 K3=1,N                                                 40
      TC(K3)=0                                                      41
  114 LOU(K,K3)=2                                                   42
      K=0                                                           43
      KT=0                                                          44
      I=1                                                           45
  115 R=0                                                           46
      L=I-1                                                         47
      CALL MASK (A,K)                                               48
      IF(A.NE.0)GOTO 128                                            49
      CALL TRACK (L,A,R)                                            50
      IF(A.NE.0) GO TO 128                                          51
  120 L=I-1                                                         52
      CALL BASE (L)                                                 53
  121 K=K+1                                                         54
      DO 122 J=1,N                                                  55
  122 LOU(K,J)=TC(J)                                                56
      IF(K.GT.500) GO TO 131                                        57
  123 L=I                                                           58
      CALL BASE (L)                                                 59
     'CALL MASK (A,K)                                               60
  124 I=I+1                                                         61
  128 IF(A) 129,130,129                                             62
  129 I=I+1                                                         63
      L=I-1                                                         64
      CALL BASE (L)                                                 65
  130 IF(I.GT.N1) GO TO 145                                         66
      GO TO 115
  131 L=L+1                                                         67
      CALL BASE (L)                                                 68
      CALL MASK (A,K)                                               69
      IF(A)139,140,139                                              70
  139 R=1                                                           71
      CALL TRACK (L,A,R)                                            72
  140 IF(L.LT.N1) GO TO 131                                         73
        DO 142 IP=1,K                                               74
  142 WRITE (9) (LOU(IP,IR),IR=1,N )                                75
      KT=KT+K                                                       76
      K=1                                                           77
      DO 143 IS=1,N                                                 78
  143 LOU(K,IS)=2                                                   79
      K=0                                                           80
      GO TO 123
  145 IF(KT.GT.100) GO TO 160                                       81
      WRITE (6,1006)                                                82
      DO 150 K1=1,K                                                 83
      WRITE (6,1003) (LOU(K1,I),I=1,N )                             84
  150 WRITE (6,1004) (LIT(I),I=1,N )                                85
      GO TO 1
  160 KT=KT+K                                                       86
      REWIND 9                                                      87
      K=100                                                         88
      WRITE (6,1006)                                                89
  162 DO 163 IP=1,K                                                 90
  163 READ (9) (LOU(IP,IR),IR=1,N )                                 91
      DO 170 K1=1,K                                                 92
      WRITE (6,1003) (LOU(K1,I),I=1,N )                             93
  170 WRITE (6,1004) (LIT(I),I=1,N )                                94
      KT=KT-K                                                       95
      IF(KT.GT.100) GOTO 162                                        96
      K=KT                                                          97
      IF(KT) 1,162,162                                              98
 1000 FORMAT(22A1)
 1001 FORMAT (4I3)
 1003 FORMAT(1H0,22I1)
 1004 FORMAT(1X,22A1)
 1006 FORMAT (1H0,'LIST OF PRIME IMPLICANTS'/)
 1020 FORMAT (22I1)
 1022 FORMAT (1H1,'TERMS OF  Y=0')
 1023 FORMAT (1H0,22I1)
 1024 FORMAT (1H0,'TERMS OF SYSTEM OF BOOLEAN EQUATIONS: Y=0')
C
C*****MODE I IN BINARY OPERATION
C*****SYSTEM-OF-BOOLEAN-EQUATIONS DISCRIMINANT DETERMINATION
C
  200 WRITE (6,1024)                                                99
      WRITE(6,1800) N,M                                            100
 1800 FORMAT(1H0,'KNOWN VARIABLES=',I4,3X,'UKNOWN VARIABLES=',I4)
      DO 205 I=1,NT                                                101
      READ (5,1020) (LOU(I,K),K=1,22)                              102
      WRITE (6,1023) (LOU(I,K),K=1,22)                             103
  205 WRITE (6,1004) (LIT(K),K=1,22)                               104
      WRITE (6,808)                                                105
      N1=2**(N+M)                                                  106
      R=1                                                          107
      DO 220 I=1,N1                                                108
      L=I-1                                                        109
      CALL BINARY (L)                                              110
      DO 210 J=1,NT                                                111
      DO 208 K=1,22                                                112
      TEMP=TC(K)+LOU(J,K)                                          113
      IF(TEMP-3) 208,210,208                                       114
  208 CONTINUE                                                     115
      L=I-1                                                        116
      CALL TRACK (L,A,R)                                           117
      GOTO 220
  210 CONTINUE                                                     118
  220 CONTINUE                                                     119
      NTF=NTF-100                                                  120
      IF(NTF.LE.0) GO TO 215                                       121
      IF(NTF-100)212,211,211                                       122
  211 NT=100                                                       123
      GO TO 200
  212 NT=NTF                                                       124
      GO TO 200
  215 WRITE (6,858)                                                125
  858 FORMAT (1H0,'DISCRIMINANT')
      CALL DISCRI (N,M)                                            126
      GOTO 1
  999 STOP
      END                                                          127
```

Figure 3—FORTRAN example (the Boolean analyzer simulator)

erated graph representing the Boolean Analyzer program. Figure 5 shows a computer generated matrix representation of the graph. It is a connection matrix with 127 columns and 127 rows. The presence of a "1" in position i, j indicates that there is an arc directed from vertex j to vertex i. The connection matrix has been triangularized such that non-triangular elements would represent program loops. The actual feedback arcs of the program loops are represented as a list of ordered pairs and shown in the upper triangular portion. Figure 6 is a list of vertex attributes. The column headings are defined in the legend at the top of the table. Figure 7 shows computer generated plots of the frequencies of execution and computation times of the vertices. They are ordered to produce a monotonically decreasing plot using a logarithmic vertical ordinate.

Now consider a programmer seeking to make use of the above analysis results. First, he must have reasonable confidence in the set of input data used to obtain vertex measurements. The plot of ordered execution times (Figure 7) then permits him to select the most important vertices as candidates for optimization and to determine their characteristics from the list of vertex attributes (Figure 6). In the present example, seven of the vertices use *95.4 percent* of the total time. These vertices are listed in Table I and show that two small subroutines account for 76 percent of the execution time. In this example further analysis indicated that a change in sequencing of the two subroutines should in itself produce considerable improvement and validating measurements are in process. When it is desired to study the detailed structure of the routines represented by vertices other analysis tools come into play. For example a question was raised in one study as to the distribution of machine language instruction types executed in a subroutine. A machine language instrumentation program was utilized to obtain the answer.

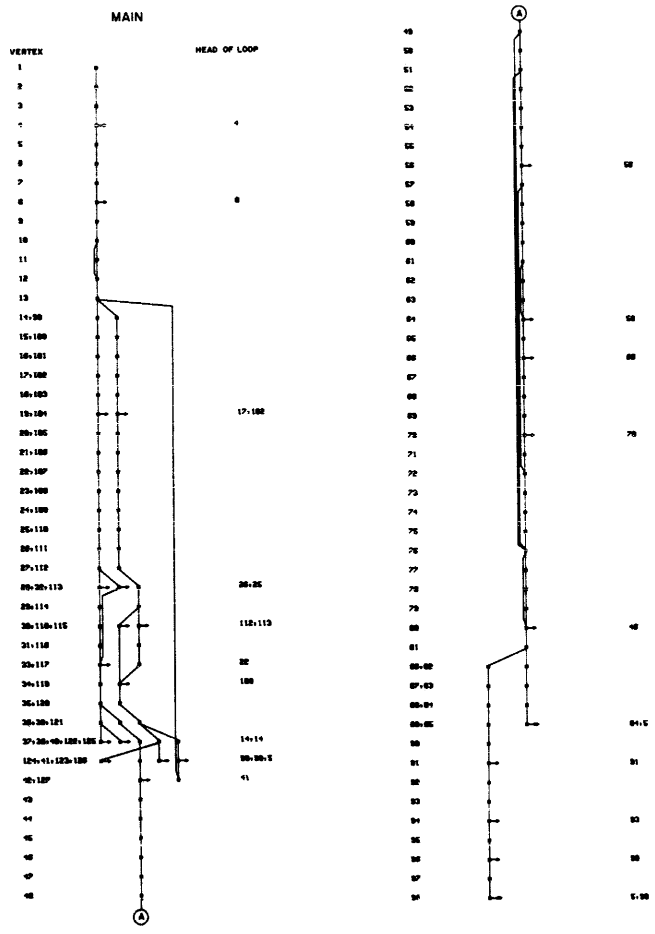


Figure 4—Flow graph of Boolean analyzer

Table I

Vertex Importance	*Vertex #*	*Vertex Name*	*Percent Execution Time*
1	48	CALL MASK (A,K)	45.3
2	23	CALL BASE (L)	15.7
3	65	CALL BASE (L)	15.7
4	26	TEMP = LOU(K,J) + JC(J)	11.7
5	27	IF (TEMP = 3) 104, 108, 104	3.4
6	84	WRITE(6, 1003) (LOU(K1,I), I = 1,N)	1.9
7	85	WRITE(6, 1004) (LIT(I), I = 1, N)	1.9

REFERENCES

1 B BUSSELL
Properties of a variable structure computer system in the solution of parabolic partial differential equations
PhD Dissertation University of California Los Angeles 1962

2 G ESTRIN R TURN
Automatic assignment of computations in a variable structure computer system
Trans IEEE EC-12 December 1963 756-773

3 D F MARTIN G ESTRIN
Experiments on models of computations and systems
Trans IEEE EC-16: February 1967 59-69

4 G R WOOD
Application of a restructurable computer system to numerical weather prediction computations
UCLA Report No 66-14 February 1966

5 E C RUSSELL
Automatic assignment of computational tasks in a variable structure computer
UCLA Report No 63-45 August 1963

6 J L BAER
Graph models of computations for computer systems
PhD Dissertation Department of Engineering University of California UCLA Report No 68-46 October 1968

7 D P BOVET
Memory allocation in computer systems
PhD Dissertation Department of Engineering University of California UCLA Report No 68-17 June 1968

8 J E RODRIGUEZ
A graph model for parallel computations
PhD Dissertation Massachusetts Institute of Technology September 1967

9 G ESTRIN D HOPKINS B COGGAN S CROCKER
Snuper computer—a computer instrumentation automaton
Proc S J C C 1967

10 M A MARIN
Investigation of the field of problems for the Boolean analyzer
PhD Dissertation Department of Engineering University of California UCLA Report No 68-28 June 1968

APPENDIX

Complete specification of the graph representation of FORTRAN IV

In the following specification for the representation of FORTRAN statements, these symbols will be used:

1. w_i represents the vertex assigned to the current statement.
2. w_{i+1} represents the next sequential statement.
3. w_n represents the statement with FORTRAN label n.
4. "(w_i, w_j)" represents an arc directed from w_i to w_j.
5. A variable will be represented in a dictionary by its attributes:

p_j—the number of bytes of storage (precision)
n_j—the name of the variable

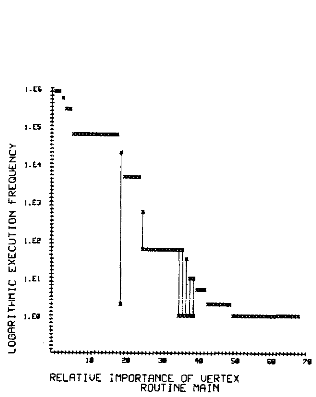
```
SEQUENTIAL CONNECTION MATRIX
    1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
  1 .
  2 1.
  3 .1.
  4 ..1.
  5 ...1.
  6 ....1.
  7 .....1.
  8 ......1.
  9 .......1.
 10 ........1.
 11 .........1.
 12 .........11.
 13 ...........1.
 14 ............1.
 15 .............1.
 16 ..............1.
 17 ...............1.
 18 ................1.
 19 .................1.
 20 ..................1.
 21 ...................1.
 22 ....................1.
 23 .....................1.
 24 ......................1.
 25 .......................1.
 26 ........................1.
 27 .........................1.
 28 ..........................1.
 29 ...........................1.
 30 ............................1.
 31 .............................1.
 32 ..........................1.....
 33 ..............................11.
 34 ................................1.
 35 .................................1.
 36 ..................................1.
 37 ...................................1.
 38 ...................................1..
 39 ..................................1....
 40 ......................................1.
 41 .......................................1.
 42 ........................................1.
 43 .........................................1.
 44 ..........................................1.
 45 ...........................................1.
 46 ............................................1.
 47 .............................................1.
 48 ..............................................1.
 49 ...............................................1.
 50 ................................................1.
 51 .................................................1.
 52 ..................................................1.
 53 ...................................................1.
 54 ....................................................1.
 55 .....................................................1.
 56 ......................................................1.
 57 .......................................................1.
 58 ........................................................1.
 59 .........................................................1.
 60 ..........................................................1.
 61 ...........................................................1.
 62 ............................................................1.
 63 .............................................................1.
 64 ............................................................1.1.
 65 ...............................................................1.
 66 ................................................................1.
 67 .................................................................1.
 68 ..................................................................1.
 69 ...................................................................1.
 70 ....................................................................1.
 71 .....................................................................1.
 72 ........................................................1.............1.
 73 .......................................................................1.
 74 ........................................................................1.
 75 .........................................................................1.
 76 ................................................1.1.......................1.
 77 ...........................................................................1.
 78 ............................................................................1.
 79 .............................................................................1.
 80 ...........................................................................1..1.
 81 ...............................................................................1.
 82 ................................................................................1.
 83 .................................................................................1.
 84 ..................................................................................1.
 85 ...................................................................................1.
 86 ................................................................................1.....
 87 .....................................................................................1.
 88 ......................................................................................1.
 89 .......................................................................................1.
 90 ........................................................................................1.
 91 .........................................................................................1.
 92 ..........................................................................................1.
 93 ...........................................................................................1.
 94 ............................................................................................1.
 95 .............................................................................................1.
 96 ..............................................................................................1.
 97 ...............................................................................................1.
 98 ................................................................................................1.
 99 ............1......................................................................................
100 ..................................................................................................1.
101 ...................................................................................................1.
102 ....................................................................................................1.
103 .....................................................................................................1.
104 ......................................................................................................1.
105 .......................................................................................................1.
106 ........................................................................................................1.
107 .........................................................................................................1.
108 ..........................................................................................................1.
109 ...........................................................................................................1.
110 ............................................................................................................1.
111 .............................................................................................................1.
112 ..............................................................................................................1.
113 ...............................................................................................................1.
114 ................................................................................................................1.
115 .................................................................................................................1.
116 ..................................................................................................................1.
117 ...................................................................................................................1.
118 .................................................................................................................1....
119 ....................................................................................................................11.
120 ......................................................................................................................1.
121 .......................................................................................................................1.
122 ........................................................................................................................1.
123 .........................................................................................................................1.
124 .........................................................................................................................1...
125 ........................................................................................................................1....
126 ............................................................................................................................1.
127 ............1...............................................................................................................1.
```

```
CYCLES IN GRAPH
HEAD  TAIL

   4     4
   8     8
  17    19
  22    33
  25    32
  26    28
  41    42
  56    56
  66    66
  70    70
  84    85
  91    91
  93    94
 102   104
 109   119
 112   118
 113   115
   5    85
   5    98
   5   126
  14    37
  14    38
  46    80
  58    64
  90    96
  90    98
  99   123
  99   124
```

Figure 5—Connection matrix of Boolean analyzer

```
                    V E R T E X   D E S C R I P T I O N   D A T A

VERTEX  EXTERNAL     LOGIC   LOCAL    PROGRAM     SINGLE-    FREQUENCY  COMPUTATION  RELATIVE
  NO.   FORMULA NO.          STORAGE  STORAGE  EXECUTION TIME              TIME      IMPORTANCE   LEVEL

  1        -1         -1      0        0          0.0          1.          0.0       127          1
  2        -2         -1      1        4      43898.2          1.      43898.2        31          2
  3        -3         -1      0        2          4.8          1.          4.8        64          3
  4         2         -1      0       10        194.9         32.       6236.8        38          4
  5         1          1      0        1      43891.0          2.      87782.0        27          5
  6        -6         -1      0        1      80593.0          2.     161186.0        25          6
  7        -7         -1      0        2          4.8          2.          9.6        53          7
  8       102         -1      0        5         12.0      20000.     240000.0        23          8
  9        -9         -1      0        2          4.8          2.          9.6        52          9
 10       -10          0      0        4          9.2          2.         18.4        48         10
 11       -11         -1      0        2          4.8          0.          0.0       126         11
 12         4          1      0        3          7.2          2.         14.4        49         12
 13       -13          0      0        3          7.0          2.         14.0        50         13
 14       100          1      0        3        177.3          1.        177.3        46         14
 15       -15         -1      0        1      80593.0          1.      80593.0        30         15
 16       -16         -1      0        2          4.8          1.          4.8        63         16
 17       -17         -1      1        7      43909.3          5.     219546.5        24         17
 18       -18         -1      1        7      80611.3          5.     403056.5        18         18
 19         5         -1      1        4      80600.2          5.     403001.0        19         19
 20       -20         -1      0        1      80593.0          1.      80593.0        29         20
 21       -21         -1      0        2          4.8          1.          4.8        62         21
 22       -22         -1      0        3          8.0      59049.     472392.0        14         22
 23       -23         -1      0        3        609.2      59049.   35972633.6         2         23
 24       -24         -1      0        2          4.8      59049.     283435.2        21         24
 25       -25         -1      0        2          4.8     285653.    1371134.4         8         25
 26       -26         -1      0       12         32.7     818355.   26760208.0         4         26
 27       -27          0      0        4          9.6     818355.    7856208.0         5         27
 28       104         -1      0        0          0.0     537254.          0.0       125         28
 29       -29         -1      0        2          4.8       4552.      21849.6        36         29
 30       -30         -1      0        3          8.0       4552.      36416.0        33         30
 31       -31         -1      0        5        223.8       4552.    1018737.6        10         31
 32       108         -1      0        1          2.0     281101.     562202.0        11         28
 33       110          1      0        0          0.0      59049.          0.0       124         32
 34       -34         -1      0        3          8.0          1.          8.0        55         33
 35       -35          0      0        4          9.2          1.          9.2        54         34
 36       -36          0      0        5         11.6          0.          0.0       123         35
 37       111         -1      0        2          4.8          0.          0.0       122         36
 38       112         -1      0        3          6.8          0.          0.0       121         36
 39       113         -1      0        3          6.8          1.          6.8        56         35
 40       -40         -1      0        2          4.8          1.          4.8        61         36
 41       -41         -1      0        5         12.0         10.        120.0        47         37
 42       114         -1      0        8         23.1         10.        231.0        45         38
 43       -43         -1      0        2          4.8          1.          4.8        60         39
 44       -44         -1      0        2          4.8          1.          4.8        59         40
 45       -45         -1      0        2          4.8          1.          4.8        58         41
 46       115          1      0        2          4.8      58995.     283176.0        22         42
 47       -47         -1      0        3          8.0      58995.     471960.0        15         43
 48       -48         -1      0        4       1762.5      58995.  103978675.2         1         44
 49       -49          0      0        4          9.2      58995.     542754.0        13         45
 50       -50         -1      0        6        225.8       4607.    1040260.6         9         46
 51       -51          0      0        4          9.2       4607.      42384.4        32         47
 52       120         -1      0        4         10.0         55.        550.0        39         48
 53       -53         -1      0        3        609.2         55.      33506.0        35         49
 54       121         -1      0        3          7.2         55.        396.0        42         50
 55       -55         -1      0        2          4.8         55.        264.0        44         51
 56       122         -1      0       11         30.3        550.      16665.0        37         52
 57       -57          0      0        4          9.2         55.        506.0        40         53
 58       131          1      0        5         11.2          0.          0.0       120         54
 59       -68         -1      0        3        609.2          0.          0.0       119         55
 60       -69         -1      0        4       1762.5          0.          0.0       118         56
 61       -70          0      0        3          6.4          0.          0.0       117         57
 62       139         -1      0        2          4.8          0.          0.0       116         58
 63       -72         -1      0        5        223.8          0.          0.0       115         59
 64       140          2      0        4          9.2          0.          0.0       114         60
 65       -74         -1      0        3          6.8          0.          0.0       113         61
 66       142         -1      1        7      80611.3          0.          0.0       112         62
 67       -76         -1      0        3          7.2          0.          0.0       111         63
 68       -77         -1      0        2          4.8          0.          0.0       110         64
 69       -78         -1      0        2          4.8          0.          0.0       109         65
 70       143         -1      0        8         23.1          0.          0.0       108         66
 71       -80         -1      0        2          4.8          0.          0.0       107         67
 72       123          1      0        3          6.8         55.        374.0        43         68
 73       -59         -1      0        3        609.2         55.      33506.0        34         69
 74       -60         -1      0        4       1762.5         55.      96937.5        26         70
 75       124         -1      0        3          7.2         55.        396.0        41         71
 76       128          2      0        3          6.4      58995.     377568.0        20         72
 77       129         -1      0        3          7.2      58994.     424756.8        17         73
 78       -64         -1      0        3          8.0      58994.     471952.0        16         74
 79       -65         -1      0        3        609.2      58994.   35939123.2         3         75
 80       130          2      0        4          9.2      58995.     542754.0        12         76
 81       145          0      0        5         11.2          1.         11.2        51         77
 82       -82         -1      0        2      80595.0          1.      80595.0        28         78
 83       -83         -1      0        2          4.8          1.          4.8        57         79
 84       -84         -1      1        7      80611.3         55.    4433620.8         6         80
 85       150         -1      1        4      80600.2         55.    4433009.6         7         81
 86       160         -1      0        4          9.2          0.          0.0       106         78
 87       -87         -1      0        0          0.0          0.          0.0       105         79
 88       -88         -1      0        2          4.8          0.          0.0       104         80
 89       -89         -1      0        1      80593.0          0.          0.0       103         81
 90       162          1      0        2          4.8          0.          0.0       102         82
 91       163         -1      1        7      43909.3          0.          0.0       101         83
 92       -92         -1      0        2          4.8          0.          0.0       100         84
 93       -93         -1      1        7      80611.3          0.          0.0        99         85
 94       170         -1      1        4      80600.2          0.          0.0        98         86
 95       -95         -1      0        3          8.0          0.          0.0        97         87
 96       -96          0      0        4          9.2          0.          0.0        96         88
 97       -97         -1      0        3          6.8          0.          0.0        95         89
 98       -98          0      0        3          6.4          0.          0.0        94         90
 99       200          1      0        1      80593.0          0.          0.0        93         14
100      -100         -1      0        1      80593.0          0.          0.0        92         15
101      -101         -1      0        2          4.8          0.          0.0        91         16
102      -102         -1      1        7      43909.3          0.          0.0        90         17
103      -103         -1      1        7      80611.3          0.          0.0        89         18
104       205         -1      1        4      80600.2          0.          0.0        88         19
105      -105         -1      0        1      80593.0          0.          0.0        87         20
106      -106         -1      0        7        186.9          0.          0.0        86         21
107      -107         -1      0        2          4.8          0.          0.0        85         22
108      -108         -1      0        2          4.8          0.          0.0        84         23
109      -109         -1      0        3          8.0          0.          0.0        83         24
110      -110         -1      0        3          4.6          0.          0.0        82         25
111      -111         -1      0        2          4.8          0.          0.0        81         26
112      -112         -1      0        2          4.8          0.          0.0        80         27
113      -113         -1      0       12         32.7          0.          0.0        79         28
114      -114          0      0        4          9.6          0.          0.0        78         29
115       208         -1      0        0          0.0          0.          0.0        77         30
116      -116         -1      0        3          8.0          0.          0.0        76         31
117      -117         -1      0        5        223.8          0.          0.0        75         32
118       210         -1      0        1          2.0          0.          0.0        74         30
119       220          1      0        0          0.0          0.          0.0        73         33
120      -120         -1      0        3          8.0          0.          0.0        72         34
121      -121          0      0        4          9.2          0.          0.0        71         35
122      -122          0      0        5         11.6          0.          0.0        70         36
123       211         -1      0        2          4.8          0.          0.0        69         37
124       212         -1      0        3          6.8          0.          0.0        68         37
125       215         -1      0        2      80595.0          0.          0.0        67         36
126      -126         -1      0        4          7.0          0.          0.0        66         37
127      -999          1      0        2          4.0          1.          4.0        65         38


TOTAL PREDICTED EXECUTION TIME =  229326976.0

TOTAL PREDICTED LOCAL STORAGE =        13

TOTAL PREDICTED PROGRAM STORAGE =     432

MAXIMUM SEQUENTIAL COMPUTATION STRING =    90
```

Figure 6—Vertex attributes of Boolean analyzer



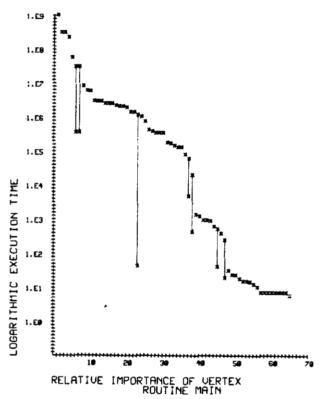


Figure 7—Vertex frequency and computation time

s_j—the number of elements (for arrays)
t_j—the type of variable

Program delimiters

Three FORTRAN statements will be processed as program delimiters

1. SUBROUTINE *name* $(a_1,a_2,a_3,\ldots,a_n)$. will be represented as vertex w_i with input data set $(a_1,a_2,a_3,\ldots,a_n)$. This input data set is given a special designation *Ip, the parameter list.*
2. *type* FUNCTION *name* $*s$ $(a_1,a_2,a_3,\ldots,a_n)$ is represented just as in (1) with the following additions. The function name is added to a list of functions to be recognized as distinct from input variables.
3. END will be represented as the terminal vertex w_z. All statements which would terminate the execution of the program will be connected to w_z. The output data set of w_z is I_p.

Specification statements

The specification statements are non-executable. They provide information about the type, precision and dimensionality of variables, together with possible sharing of storage between variables.

1. *type* $*s\ a_1\ *s_1\ (k_1)/x_1/,\ a_2\ *s_2\ (k_2)/x_2/,\ldots,a_3\ *s_n\ (k_n)/x_n/$. Each variable in a type statement will be entered in the dictionary according to the following conventions:
   a. name (n_i) is stored directly
   b. type (t_i) is taken from the statement type (INTEGER, REAL, LOGICAL, COMLEX)
   c. precision (p_i) is either determined from the statement (if included) or defaults to the following values:

type	t_i	P_i
INTEGER	1	4
REAL	2	4
LOGICAL	3	4
COMPLEX	4	8

   d. size s_i is computed as product of k_i expression.
2. DIMENSION a_1 (k_1), a_2, $(k_2),\ldots,$ a_n (k_n) Dimensioned variables will be entered into the dictionary (if not previously defined by a type of COMMON statement) and the size (s_i) is computed from k_i.
3. COMMON $/r_1/a_1\ (k_1),\ a_2\ (k_2)\ldots/r_2/a_3\ (k_3),\ldots$ Common variables are entered into the dictionary (if not previously defined by a type or dimension statement) and the size (s_i) is computed from k_i. In addition, all variables in COMMON blocks are placed in a special set, the *common list* I_c.
4. EQUIVALENCE $(a_1, a_2, a_3, \ldots), (a_4, a_5, \ldots), \ldots$ The equivalence statement has two effects on the program description. First, the storage allocation for defined variables is altered. If the variables in an equivalence class are not completely overlapped then some modification of the calculation of the amount of storage is required. This is especially important when such variables lie in COMMON blocks. If storage allocation were the only reason for using equivalence statements, they could be ignored in this analysis. However, the values of variables may be referenced by any name which happens to be assigned to the cell. Thus, it is imperative to include all "aliases" when preparing the input and output data sets. A well-written program for a single processor may obscure potential parallelism due to storage sharing.

Control statements

The control statements are the source of the majority of logical control arcs. The FORTRAN statements included in this group are the GO TO statements (unconditional, computed and assigned), the IF statement (arithmetic and logical), the DO, CONTINUE, PAUSE, RETURN, CALL, and STOP statements.

1. GO TO statements
   a. GO TO *n*
      the unconditional transfer statement pro-

duces an arc of the form (w_{i-1},w_n). This statement does not result in a new vertex.

b. GO TO $(n_1,n_2, \ldots ,n_k)$ j
the computed GO TO statement is represented by a vertex (w_i) with multiple out-branchings (w_i,w_{n_1}), $(w_i w_{n_2}), \ldots$, (w_i,w_{n_k}). w_i has exclusive-OR output logic. $I_i = \{j\}$

c. GO TO j, $(n_1,n_2, \ldots ,n_k)$
the assigned GO TO statement is represented exactly the same as (b)

2. IF statements

a. IF (e) n_1,n_2,n_3
The arithmetic if statement is represented by a vertex w_i with exclusive or output logic and out-branching arcs (w_i,w_{n_1}), $(w_i,w_{n_2}),(w_i,w_{n_3})$. The input data set consists of all variables referenced in the arithmetic expression "e."

b. IF (e) s
the logical if statement is represented by a vertex w_i with exclusive or output logic and outbranching arcs (w_i,w_{i+1}), and (w_i,w_{i+2}). The input data set consists of all variables referenced in the logical expression "e." Statement "s" is treated as a separate vertex (w_{i+1}).

3. DO $n\ i = m_1,m_2,m_3$
The DO statement is represented by a vertex w_i with $I_i = \{m_1,m_2,m_3\}$ and $O_i = \{i\}$. Arcs produced are (w_i,w_{i+1}) and (w_n,w_{i+1}). The latter arc is a feedback connection for the loop.

4. CONTINUE
The CONTINUE statement is represented by a vertex w_i with $I_i = \Phi$, $O_i = \Phi$ and out-branching arc (w_i,w_{i+1}).

5. PAUSE
The PAUSE statement is represented by a vertex w_i with $I_i = \Phi$ $O_i = \Phi$ and out-branching arc (w_i,w_{i+1}).

6. RETURN
The RETURN statement produces an arc of the form (w_{i-1},w_z). This statement does not result in a new vertex. (w_z is the pseudo-terminus of the program.)

7. STOP
The STOP statement is represented just the same as a return.

8. CALL *name* $(a_1,a_2, \ldots a_n)$
The CALL statement is represented by a vertex w_i. In the usual case there is a single outbranching (w_i,w_{i+1}). However, it is possible to specify other return paths as parameters to the subroutine in the form CALL *name* $(a_1,\ldots$ & $a_{n1} \ldots$ & $a_{n2})$ in which $\&a_{n1}$ and $\&a_{n2}$ represent statement numbers in the calling program. In this case, multiple outbranching arcs are generated (w_i,w_{i+1}), (w_i,w_{n_1}), $w_i,w_{n_2}), \ldots$ with output logic exclusive or. In the absence of further information, all parameters are assumed to be members of both I_i and O_i. Also any COMMON variables must be considered part of both sets. Thus

$$\{I_i = a_1,a_2, \ldots ,a_n,I_c\}$$
$$\{O_i = a_1,a_2, \ldots ,a_n,I_c\}$$

Input/output statements

There are five I/O statements which will be included in this discussion: READ, WRITE, END FILE, REWIND and BACKSPACE.

1. READ (*a, b*, END = n_1, ERR = n_2) *list*
This statement is represented by a vertex w_i with input data set $I_i = \{a,m_1,m_2 \ldots\}$ where $\{m_1,m_2 \ldots\}$ represent any loop limits in the *list*. $O_i = \{\text{list}\}$. The exclusive-outbranching arcs are (w_i,w_{i+1}) and optional arcs (w_i,w_{n_1}) and (w_i,w_{n_2}). Other variations on the READ statement are processed similarly.

2. WRITE *(a,b) list*
This statement is represented by a vertex w_i with input data set $I_i = \{a, m_1, m_2, \ldots,$ *list*$\}$ where $\{m_1,m_2, \ldots\}$ represent any loop limits in *list* and *list* represents the variables to be output. $O_i = \emptyset$ and there is a single outbranching (w_i,w_{i+1}).

3. ENDFILE a
REWIND a
BACKSPACE a
The structure of these three statements is the same. Each is represented by a vertex w_i with $I_i = \{a\}$, $O_i = \Phi$ and a single outbranching arc (w_i,w_{i+1}).

Arithmetic statements

The arithmetic statement, most generally of the form "a = e" is represented by a single vertex, w_i, with $O_i = \{a\}$ and $I_i = \{k_1, \ldots ,k_n\ b_1, \ldots ,b_n\}$ where the k's are subscripts used anywhere in the statement and the b's are variables in the arithmetic expression "e". A single outbranching arc (w_i,w_{i+1}) is produced.

Software measurements and their influence upon machine language design*

by L. PRESSER and M. A. MELKANOFF

University of California
Los Angeles, California

INTRODUCTION

At present, software development is responsible for a large part of the total cost of computer systems. A segment of this cost is traceable to the development of programming language (e.g., FORTRAN, ALGOL, PL/I) translators which constitute major components of any software package.

Albeit machine languages have evolved through the various computer generations, the machine language of present day conventional computers is still too elementary (e.g., string manipulation operations require considerable set-up time) for a simple and concise implementation of translators and for carrying out the actual translation process in a manner which makes effective use of the available hardware.

In order for future information processing systems to make effective use of the available technology, an appropriate evaluation[1] of the complete operation of present information processing systems is needed. In particular, we are interested here in obtaining information concerning the manner in which the translation of programming languages may influence the machine language design of future computers, the objectives being the production of a well-balanced integrated design and the simplification of the translator writing task.

In this paper we shall discuss a specific programming language translator and a measurement control center, and associated software artifact, incorporated into the translator at implementation time in order to gather, optionally, information about the translation process. Utilizing a number of benchmark programs measurements are collected to determine the relative effort spent in the various sections of the translation process by this particular translator. The type of information so obtained contributes to a better understanding of how the translation of programming languages may influence machine language design.

Translation technique

The system described here is essentially the META5 translator writing system.[2] This system has been extended and implemented at UCLA on both the IBM SYSTEM/360 and the SDS SIGMA 7 computers. Henceforth, for purposes of this paper, we may view this translator writing system as a translator for the META5 language.

The META5 system employs a two-pass technique. The first pass transforms, interpretively, a META5 program into a pseudo-code or intermediate language (i.e., between machine and programming language) representation. The second pass interpretively executes the pseudo-code on the associated pseudo-machine.

Essentially, given a language, we refer to a computer and associated routines which behave as a pseudo-machine for which the given language is the machine language as an interpreter of that language.

Through a bootstrapping approach a pseudo-code version of the program which transforms META5 programs into pseudo-code programs was obtained.* Execution of this program, on the pseudo-machine, is what constitutes the first pass of the overall process.

* The work was supported by Atomic Energy Commission AT(11-1) Gen 10 Proj. 14; Advanced Research Projects Agency, SD 184; Information Systems Division Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government.

* First, Oppenheim's[2] implementation was bootstrapped to the IBM SYSTEM/360[3]; obtaining a META5 to PL/I translator in PL/I. The next bootstrapping produced a META5 to pseudo-code translator, in pseudo-code, on the SYSTEM/360.[4]

Measurements

The primary type of measurements[5,6] in which we are interested are:

1. Event statistics (in particular as required to determine that part of the total translation effort contributed by the various sections of a translator):
   a. Time
   b. Frequency
2. Employment of resources (storage space in particular)
3. Interaction with the Operating System
4. Structure of translated programs

The measurement process can be carried out to varying degrees.[6] Hence, the control information supplied by the Operating System to the translator may contain an indication of the amount of effort to be dedicated to the collection of measurement data.

At the completion of the translation process (in the case of META5 at the completion of each pass), the collected measurement information is processed, formatted, and recorded on the appropriate data set (organized collection of related information) which is then made available to the Operating System. This is done in the same manner as for the other various forms of output.

When a translator is being specified it is possible to indicate the type of measurements desired and the maximum cost one is willing to pay for the obtainment of the measurements. This cost may be stipulated as a percentage of the time employed by the measurement-free translation process as far as timing specifications is concerned, and as a percentage of the resources (storage space in particular) employed by the translator when it contains no measurement artifact, as far as the utilization of resources is concerned.

It is to be emphasized that the introduction of measurement artifact for collecting most of the type of measurements in which we are interested should be an inexpensive undertaking if this task is taken into consideration during the design of the translator; moreso if the translator is a well-organized and well-designed translator. What may involve some cost is the processing and formatting of the measurement data obtained. However, this is considered as a separate task subsequent to the translation process proper.

A special unit has been incorporated into the control section of the META5 translator with the sole purpose of controlling the collection of information about the translation process. This unit controls simple software measurement artifact which has been introduced into the system.

From the programmer's point of view, the measurement instrumentation consists of a set of possible measurements (i.e., MEASURE1, MEASURE2, . . ., MEASUREI, . . .where I > 0) where each of MEASUREI gathers information on a specific set of events and each can be turned ON/OFF (for either the first or the second pass) at the META5 language level by appropriate control commands.

In the META5 case the measurements (only the collection of event statistics is discussed in the sequel) are obtained as follows. The META5 implementation consists, in part, of a pseudo-instruction decoder and a set of routines. Each time a pseudo-instruction is decoded a series of calls is made to the appropriate routines in order to execute the pseudo-instruction. Most of the measurement artifact has been introduced at the decoder level. Each time a routine is called the measurement artifact increments a frequency counter associated with that routine and records the clock; when that routine returns, the clock is recorded again and the elapsed time added to a time counter associated with the routine. These measurements provide statistics on the various pseudo-instructions.

Measurement artifact has been introduced at various other points (very few outside decoder) in order to record the activity associated with certain specific events. For instance, the clock is recorded right before and just after an I/O operation and the elapsed time recorded and associated frequency counter incremented. Also, measurements are gathered on the total time spent in such events as the pseudo-code loading process. At the end of a pass the collected measurement data are processed, formatted, and output in an easy-to-read form. Figure 1 displays a structural diagram of the META5 system which includes an indication of the location of its measurement artifact.

The perturbation introduced into the translation process by the measurement activity is relatively minor, as shown below.

Measurement data

The results of a number of benchmark measurements are tabulated in Tables I through III.

The benchmark programs utilized were:

META5 to pseudo-code.	Denotes the translation (first pass) of META5 (which is written in META5) to pseudo-code.
ECR to pseudo-code.	Denotes the translation (first pass) of a META5 program to pseudo-code. This program analyzes

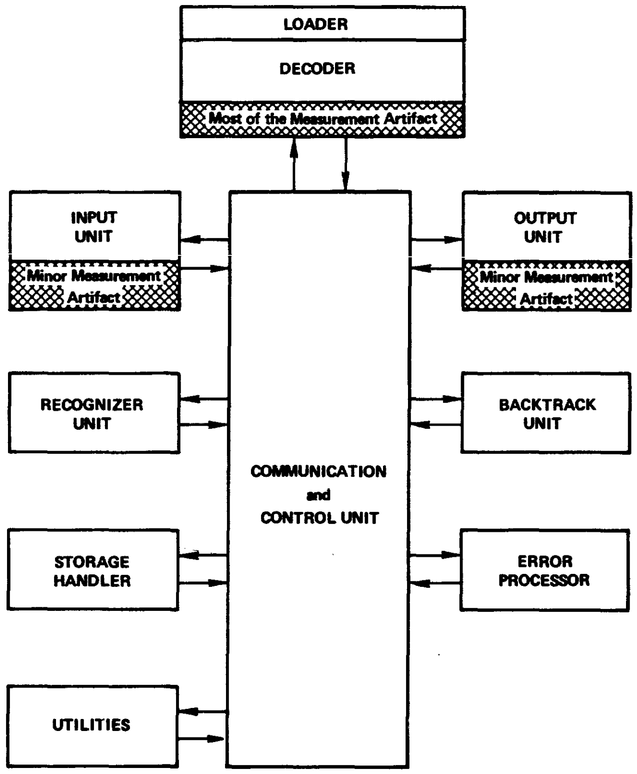

LOADER
DECODER
Most of the Measurement Artifact
INPUT UNIT
Minor Measurement Artifact
OUTPUT UNIT
Minor Measurement Artifact
RECOGNIZER UNIT
BACKTRACK UNIT
COMMUNICATION and CONTROL UNIT
STORAGE HANDLER
ERROR PROCESSOR
UTILITIES

Figure 1—Structural diagram of the META5 system

	FORTRAN programs for purposes of parallel execution.[4]
RM to pseudo-code.	Denotes the translation (first pass) of a META5 program to pseudo-code. This program accepts as input certain logical equations and produces as output wire-list information.[3]
CK to pseudo-code.	Denotes the translation (first pass) of a META5 program to pseudo-code. This program reformats META5 programs into a specified format.[7]
Execution of ECR.	Denotes the execution (second pass) of the ECR program. The input data used was a FORTRAN program which simulates a Boolean Analyzer.[8]
Execution of RM.	Denotes the execution (second pass) of the RM program. The input data used was a small set of logic equations.

Tables I through III are self-explanatory; however, some of the salient points should be noted:

1. Status restoring, in the backtracking process, represents the most time-consuming (about 29 percent) section of a META5 translation (first pass) process.
2. Inputting and outputting together represent the second most time-consuming (about 25 percent) section of a META5 translation (first pass) process.
3. String comparisons and table look-ups together represent the third most time-consuming (about 20 percent) section of a META5 translation (first pass) process.
4. Arithmetic operations constitute a relatively minor part (about 2 percent) of a META5 translation (first pass) process.
5. The effect of the measurement artifact is relatively minor ($\leq$ 6 percent).
6. The system is printer bound when source listings are requested. If no listings are required the translator is compute bound.
7. The backtrack stack requires many levels (37 levels for the benchmark programs used here).

Finally, the following definitions may clarify the headings appearing in Table III.

The inputter is a unit which obtains, decodes, and places in a repository buffer, elements from the input stream. It permits backtracking the head of the input stream if required, and allows the optional advancing of the head of the input stream past certain characters

Table I

META5 System — Absolute Times

	Time spent in interpreter (excluding pseudo-code loading time) in ms.					
	META5 to pseudo-code	ECR to pseudo-code	RM to pseudo-code	CK to pseudo-code	Execution of ECR	Execution of RM
Listing of source program and gathering of measurements	167,014	137,594	123,648	61,930	133,758	19,486
Listing of source program and no gathering of measurements . .	157,978	131,082	117,646	58,124	126,548	18,558
No listing of source program and gathering of measurements . .	148,722	117,702	108,324	53,744	122,880	17,072

Table II

META5 System — General Measurements

	META5 to pseudo-code	ECR to pseudo-code	RM to pseudo-code	CK to pseudo-code	Execution of ECR	Execution of RM
% increase in interpretive execution time due to listing of source program	12	17	14	15	9	14
% increase in interpretive execution time due to gathering of measurements	6	5	5	6	5	5
Total number of pseudo-instructions executed	224,591	173,383	159,602	78,419	193,925	26,234
No. of records in source program input	253	318	231	121	154	37
No. of records output	704	575	485	285	708	157
Maximum height of the back-track stack	37	30	31	29	20	14
Time spent loading pseudo-code in milliseconds	27,446	27,446	27,442	27,410	21,644	18,308

Table III

META5 System — Detailed Measurements

	Percentage (See Note 1)											
	Lexical Analysis (See Note 2)		Backtracking Management			String comparisons (See Note 3)	Table look-ups	Write operations into pseudo-machine memory	Read operations from pseudo-machine memory	Arithmetic operations	Outputter	Other
	Inputter	Other	Status recording	Status restoring	Other							
META5 to pseudo-code	14	5	5	31	1	9	9	4	8	2	12	—
ECR to pseudo-code	13	6	4	26	1	9	14	4	8	2	11	—
RM to pseudo-code	15	6	4	32	1	11	9	3	8	2	11	—
CK to pseudo-code	15	5	4	28	1	10	10	4	9	2	12	—
Execution of ECR	5	3	4	8	1	3	12	4	17	11	16	16
Execution of RM	8	4	5	16	1	4	4	6	11	1	23	17

Note 1: No listing of source program and gathering of measurements.

Note 2: The figures listed under lexical analysis represent a low approximation (no symbol descriptors are emitted) to the usual definition of lexical analysis[9].

Note 3: These are comparisons with the head of the input stream; thus this percentage includes implicit calls to the inputter. As a consequence, part of the percentage listed under this column is counted twice, here and under the column labeled Inputter.

(e.g., blanks). The inputter is a component of the lexical analyzer.

The outputter is a unit which serves as a repository buffer for the output stream and allows the specification of output record format. It also permits backtracking the head of the output stream if required.

CONCLUSIONS

This paper suggests the stipulation of a measurement control center and associated artifact as a regular feature of programming language translator specifications. Experience shows that the introduction of the software measurement artifact required for the collection of the type of measurement data discussed here is a simple and inexpensive task if carried out at translator design time. Furthermore, the sample data presented here indicate that the optional measurement activity places a minor burden on the translation process.

The information to be gathered with such systems can have great influence on future computer architecture, especially machine language design, and in the simplification of the translator writing process. In addition, this type of data aid in the determination of the direction of effort for future translator improvement and in the evaluation of newer translator versions.

It should be emphasized that the results presented here were obtained on a single translator and specifically on the META5 SIGMA 7 implementation with which the authors were involved. Thus, even though there are no reasons to consider the implementation in question atypical, the data must be viewed in proper perspective.

It is also worth noting that META5, as implemented, is an interpretive system. If it were a compiling system with, for example, elaborate optimization as a goal, the relative percentages would show some changes.

These measurements substantiate the fact that the translation of programming languages is not an arithmetic-oriented problem but rather a string-manipulation problem. Hence, as far as the translation of programming languages is concerned, a machine language based on string manipulations rather than arithmetic operations, should allow the effective, simple, and concise implementation of programming language translators. These considerations suggest some very interesting nonconventional machine organizations.

As far as the determination of syntactic structure is concerned, the high cost of backtracking serves as strong justification for the considerable effort spent by many researchers on precedence techniques[6] which do not require the backtrack mechanism.

Finally, it is indeed worth mentioning some simple hardware generally absent from present-day computers, which would facilitate the coding of software measurement artifact. To wit:

1. A variety of clocks, especially very fast clocks (of the order of an instruction cycle) with a very fast access time, as well as clock manipulation instructions. Another type of clock which would be useful would be clocks which accumulate time intervals.
2. A number of counters, as well as counter manipulation instructions.

There is no reason why these features could not be incorporated into a system at system design time in the same manner, for instance, that hardware facilities have been added to the newer generation of computers for purposes of error detection and correction.

REFERENCES

1 G ESTRIN et al
Snuper computer—A computer instrumentation automaton
Proc S J C C 1967

2 D K OPPENHEIM
The META5 language and system
System Development Corp Report TM-2396 1966

3 R MANDELL
Private communication

4 E C RUSSELL
Private communication

5 G ESTRIN
Measurement definitions and discussions
Unpublished paper

6 L PRESSER
The structure, specification, and evaluation of translators and translator writing systems
PhD dissertation Dept of Engineering University of California Los Angeles 1968

7 C KLINE
Private communication

8 M A MARIN
Investigation of the field of problems for the Boolean analyzer
PhD dissertation Dept of Engineering University of California Los Angeles 1968

9 T E CHEATHAM JR
The theory and construction of compilers
Computer Associates Inc Report CA-6606-0111 1966

More on simulation languages and design methodology for computer systems*

by DAVID L. PARNAS

Carnegie-Mellon University
Pittsburgh, Pennsylvania

INTRODUCTION

In an earlier paper[1] we attempted to set forth (1) a design methodology for computer systems which made heavy use of simulation and (2) a simulation language intended to facilitate the use of the design methodology presented. The basic justification for the design methodology presented an old precept from engineering design: a problem must be defined before it is solved. The result was a methodology which laid great stress on specifying the behavior of a system or a component in a system before producing the design. The simulation language, SODAS, was designed to allow a design to proceed in a hierarchical way, treating any system as a set of components, specifying the behavior of those components, then treating the components themselves as systems. By means of the SODAS language it was to be possible to evaluate the design at any stage in its development without excess effort.

One of the most fruitful results of the publishing of "SODAS. . . . ."[1] has been a number of useful discussions on design methodology and simulation with other workers. Probably the most fruitful of these discussions has been with Brian Randell, who had been interested in similar problems, but with a somewhat different emphasis and result.[2] A major result of those discussions has been the realization that while the basic design methodology described in "SODAS. . . . ."[1] is most general and should apply to all sorts of computer system design problems, the specific characterization of it given in "SODAS. . . . ."[1] itself fails to meet such application requirements.

An examination of the work leading to and motivating SODAS shows that all the motivating examples fall into a rather restricted class of hardware modules. That class can be roughly characterized as single level*, involving no interpretation of programs or even microprograms. It is not surprising then that we find that the particular description of the design methodology given in "SODAS. . . . ."[1] and the structure of SODAS itself are fairly closely restricted to that class of system.

It is our purpose in this paper to explore a somewhat less restricted expression of our basic design methodology in an attempt to extend it beyond the limited confines covered by "SODAS. . . . .".[1] In particular, we expect to explore the design of computer systems consisting of at least two (and often more) levels of hardware and software. We shall refer to our system class as the design of Operating Computer Systems (OCS), since it usually includes both hardware and the software known as the operating system for the hardware. It is also our hope to take into account the concepts about multilevel simulation given in Zurcher and Randell,[2] working them into the design of a sim-

* This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F 44620-67-C-0058) and is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

* It is necessary to note here that we are deviating from the way that level is used in "SODAS. . . . ."[1] to a usage that is more consistent with Zurcher and Randell.[2] In "SODAS. . . . .",[1] level referred to level of detail, i.e., the number of levels of definition design completed in a state of partial design. The use of level here is somewhat more complex and will be defined more precisely in the text.

ulation language which is more broadly applicable than SODAS.**

The principal difference between the class of systems discussed in "SODAS. . . . ."[1] and the class that we are now interested in arises from the existence of a program known as the operating system, which should be considered an integral part of the design problem. Early computers were designed with no thought of an operating system, simply because they were to be run without an operating system, As the need for an operating system became apparent, these were designed separately from the fixed piece of hardware and had as their aim efficient and convenient use of that piece of hardware.

The premise that we should proceed by specifying the behavior of a system before designing its components implies that we can no longer look at an operating system as an item to be placed on a previously designed piece of hardware. The actual design should begin with a specification of the overall behavior of the hardware/software combination. It continues by dividing the system into components and they, in turn, are designed with little or no attention to the question of what will be hardware and what will be software until very late in the design.

In the type of system that SODAS deals with, the nature of the system building block was clear; each hardware module is composed of smaller or lower level hardware modules until one gets to the realm of logic elements. For systems of hardware and software (OCS) we must use as a component or building block a unit which will allow us to postpone the decisions about the hardware/software tradeoff. The unit we have selected is the "sequential process", a rather fuzzy concept which has been discussed elsewhere.[3,4,5] For our present purposes we note that a sequential process is a fully ordered set of events in an operating computing system and may be performed either by hardware or software. By building a design as a set of such processes we can postpone the hardware/software decision as long as desired.

Difficulties with SODAS

In the following section we shall attempt to discuss some of the difficulties that the existence of hardware and software as part of a system design introduces into any attempt to use SODAS for such designs.

1. The existence of interconnectors at a lower level (further progress in the design) that did not exist at higher levels (earlier in the design).

   In the hardware modules any wires that connect parts of one subsystem with parts of another subsystem must explicitly connect those two subsystems. As a rule (though not invariable) all significant interconnectors between two components are specified at the time that the functions of those components are specified. No new interconnections show up as the design progresses. In operating systems this is not the case. It is reasonable and desirable that at certain stages in a design the several sequential processes will be described as entirely independent of each other except for certain explicit attempts at communication. At a later stage in the design we will note that these processes have an implicit intercommunication because of resource sharing. In SODAS with its requirement of explicit interconnectors, this would require that the descriptions at the upper level be rewritten. This violates a design criterion for SODAS—the use of SODAS should require no extra effort
2. The problem of hidden resources.

   At an early stage in the design it will be reasonable to assume that all of a certain class of resource that will ever be required by a process will always be available. At a later stage we decide that for cost reasons it will not. The process will then go to use that resource, find that it is not there and wait until it is available. SODAS will not permit us to add this level of detail without rewriting the description we produced at the more abstract level.
3. Communication through global variables.

   The pretense that all communication is through an explicit set of wires or variables becomes unreasonably restrictive when talking about software operating systems. The simulation language and design methodology must recognize the communication of processes through global variables and files as well as more subtle means of communication (e.g., changing the instruction counter of a process or changing its code).
4. Expansion of time points to time intervals.

   At early days in the design of an OCS certain sequences of computations can be considered to be single events occupying a "point" in time

** We have chosen in this paper to talk not about languages, but about the characteristics or features of languages. We will not at any stage in this paper give a syntax of the language, or even a part of that syntax. Rather, we shall talk exclusively about properties that a language must have for the purposes outlined here. In part this is due to a feeling that the syntax of the language is an irrelevant detail as well as the fact that the syntax of the language is far from frozen at this stage.

and separated by a specified period during which nothing happens. It is assumed that the event cannot be interrupted and that no time passes except between events. At later stages in the design it often becomes necessary to recognize that time passes during an event since conditions not considered at earlier stages of the design may cause the process to be interrupted during an event. This in itself could conceivably be handled within SODAS, or other simulator, but there is no facility that would allow the simulation system to determine the simulated time of this interruption of the event unless we force the programmer to insert timing throughout. The latter requires either recognition of the problems of a later stage of design at too early a stage or substantial modifications to a description which is actually perfectly valid for the level at which it is written.

Multi-level modeling—two views

Randell and Zurcher[2] have presented a view of system design and simulation which, on the surface, is quite similar to that presented in "SODAS. . . . ."[1] Both papers discuss the value of being able to have a simulation model which simultaneously includes descriptions at many levels of detail which are interacting. Close inspection, however, of the two papers indicates that the word 'level' is being used in two different senses. The design process discussed in "SODAS. . . . ."[1] can be viewed as repeated functional decomposition. Each unit is decomposed into a sub-unit, each with a distinct function. When "SODAS. . . . ."[1] speaks of 'levels of detail' it is referring to the number of times this decomposition has been carried out in a given unit. Randell[2] sees a different direction of progress for a design, a given functional unit will initially be described as an abstraction from reality. Certain facts about its implementation are initially ignored, while certain design features are being explored. As one recognizes and deals with these facts of life which were initially ignored, one proceeds from a high level of abstraction to a lower level of abstraction. Often in doing a functional decomposition one is simply introducing functional units which are needed to proceed from an abstract description of a component to one which enables the component to exist in a restricted real environment. In proceeding to a lower level of abstraction one is often simply specifying the nature of certain components of the given component. Often, however, the two notions of level are quite different. In functional decomposition the functional units specified are always sub-units of the unit we are decomposing. That unit is always viewed as self contained; in contrast, it is common in going to a lower level of abstraction to recognize sub-units which must be viewed as shared by other components. Thus we are not simply decomposing a given component into its sub-components. Often, too, proceeding to a lower level of abstraction does not involve any functional decomposition at all; it may merely involve taking into account certain external influences on a given component. On the other hand, functional decomposition does not always involve proceeding to a lower level of abstraction. To avoid confusion between these two concepts, we shall hereafter use a "level" to refer to *levels of abstraction.* "*Ex-tent of functional decomposition*" will be referred to explicitly, where necessary, by that rather unwieldy phrase.

Both extending functional decomposition and lowering the level of abstraction are consistent with the basic precept of our design methodology; both involve proceeding from a specification of desired behavior to a means of achieving it.

The nature of distinct levels in OCS

To date we have left "level" as a fuzzily defined concept. It is necessary now that we determine the nature of levels of abstraction within a complete system, i.e., that we answer the question: Under what circumstances must we consider two actions in a computer system to occur at different levels—or—how do we justify the statement that two subroutines within the system deal with it at a certain level?

A fairly easy answer is that there is no concrete realization of the term level within the system. That the division between levels of abstraction is entirely in the mind of the analyst or designer, who determines what is in the top level by what he chooses to consider first. This rather easy answer, however, ignores a regularity that can be observed if one studies a number of examples of systems so divided into levels.

It appears to be the case that if we have described a set of processes at some level when we proceed to the lower level, we are either specifying or modifying the specification of the interpreters of those processes previously described. In other words, a top level description of a system is a description of a set of processes with an assumed interpreter, that interpreter being essentially passive and accomplishing the actions specified by the process description as directly as possible. When we move down to a lower level, we begin to assign more complex functions to the interpreter, e.g., managing the resources being used by the process and suspending action on the process should insufficient resources be available.

What SODAS needs

SODAS was designed under the assumption that all the component descriptions and/or process descriptions could be run using the same fixed interpreter. There was no provision for making any substantial alterations to the behavior of the interpreter or for including new ones as part of the design process. Thus SODAS HAS AN INHERENT LIMITATION TO SINGLE LEVEL SYSTEMS.

Each of the difficulties pointed out earlier can be *neatly* avoided by providing the ability to (a) make certain changes in the operation of an interpreter supplied as part of the system and (b) to provide for the possibility of some simulated processes being the interpreters for others.

SOCS

Having explored a little further the nature of the design process for an OCS, we shall now look at the design of a simulator for use in designing an OCS (SOCS). We can first review the features that we found missing from SODAS.

1. Communication through global variables rather than interconnectors.
2. A flexible interpreter.
3. Ability to simulate a process which is being interpreted by other processes being simulated.
4. Integration of the process describing languages with the network description language.*

We may also list the features that we found in SODAS that were missing in such process oriented languages as SIMULA.[4]

1. Ability to have a process that consists of a set of processes, e.g., recursive structure.
2. Ability to handle difficult cases of simultaneous events.
3. Ability to handle structural descriptions of hardware.
4. The "wait until" or monitoring feature found in SOL, proposed for SODAS, but missing in SIMULA.

It is not possible at this point to give a detailed description of the SOCS language. It is clear that it will be somewhat reminiscent of SIMULA with extra constructs to provide the features that we have discussed. It is desirable that the language be such that a SIMULA program could be run without *substantial* changes though we would not object to extensive surface changes. Although not essential, it will probably prove convenient to provide constructions which allow processes to be dealt with as SOL facilities or stores. The sub-languages and connection concept will be carried over from SODAS, though with a somewhat subdued role as it will be possible but not necessary to leave the SOCS language to describe a process (SODAS allowed no primitive process to be described in SODAS itself). At present the problem of simultaneous events looks somewhat less central than it did earlier and will be provided as an optional, *run time* resolver rather than the compile time solution planned for SODAS.[6]

* This refers to the fact that SODAS was a language for connecting components described in other languages rather than a language system.

On following a design philosophy

It is fairly easy to talk about designing from specifications to plans, outside in, top down, or from process to processor. The way that a design should progress is, at least at a certain level of generality, agreed upon. It is quite a different matter to actually carry out a design in that way when dealing with an OCS.

There are two serious difficulties that are encountered should one try to follow this philosophy. The first of these is an inability to really get started, to begin to write down concrete information about the design. One finds oneself in the position of trying to write a program for a machine of unknown characteristics. There is no starting point, no structure to build upon. We are used to designing processes or programs by starting with a very specific tool and looking for a way to take this rather limited tool and get around the limitations. Without the limited tool, with the limitations that we have become accustomed to suddenly gone, we feel lost; the space of possible first steps has become so large that we cannot make the decision.

This difficulty can be somewhat alleviated by providing a higher level language in which to write. We again have restrictions; the syntax and semantics of that language, and the space is again reasonably small. (The space, however, is nowhere near as small as that provided by a machine language and there are very good machine language programmers who still cannot readily get started when faced with a higher level language to write in. This is not a widespread phenomena, however.)

A difficulty which is expected by many people with whom I have discussed the philosophy is the problem of duplicate work. There have been many system designs in which simulation played a part, but it was clearly separate from design work. There was a separate group doing the simulation and it required either extra money or a slower effort by the design group. With talk of really extensive reliance on simulation, people with such experience see a project in which everything is done twice—once for the simulator and once for the actual system. While some might argue that even such double effort would be worth it if a well designed system were to result, I wish to argue that such double effort can and should be avoided. The code written for the simulator must eventually become part of the running system. If we write a description for a process in some higher level language, such as SOCS, and later specify and design an interpreter for that process, the higher level language will have to be translated into a pseudo-code that is the language of the simulated interpreter. The translator, if the translation is done by a program, and the translated code should both eventually become a part of the system. If, for example, the interpreter is designed directly as hardware, the code is now machine code; if the interpreter becomes a program on some other hardware, it will continue to operate on the same code. The effort involved in producing it for the simulation is not wasted; the product becomes a part of the system that is eventually produced.

Another important aspect of avoiding duplicate work is that it not be necessary to rewrite a description correct at the level of detail at which it was written, because we are progressing further in the design˙ (We will never avoid rewriting descriptions when we make design changes.) This can be avoided to a very large extent by our ability to modify the behavior of processes, without altering their description, by altering their interpretation or interpreter. The original description thus remains both for purposes of explanation and as a specification of what is expected.

The second real difficulty is encountered at the end of the design process, and has been reported to me by Brian Randell. Randell had been following the design process outlined using a facility for multi-level modeling implemented using FORTRAN running under OS/360. When he reached the point that he felt that the design was complete, he found that it could not become the system, since it still contained many points of dependency on FORTRAN, OS, and the /360 itself. A design done in the way we are discussing cannot be considered complete until the only processes being interpreted by the system interpreter are those simulating the hardware. All other processes must be interpreted by the simulated hardware or by processes being interpreted by the simulated hardware processes, etc. A system like Randell's which was not specifically designed to allow one process to interpret another, cannot allow the design to proceed to that stage.

Future work

The alert reader will note that while we have been talking about design we have not been designing. It is our intention to design a version of SOCS which can be quickly implemented. We intend to go through a complete design of a system of interest in itself, keeping a protocol of the way that the design progresses. In this way we hope to determine just how accurate our picture of the design process is and further, in what ways SOCS will have to be altered in order to keep from interfering with or distorting the design process. We feel that after using the rudimentary system in a real design, we will have a much better picture of the final version of SOCS.

ACKNOWLEDGMENTS

I wish to acknowledge most gratefully the many hours of discussion that I have had with Brian Randell; each of those hours has improved my understanding of the problems dealt with in this paper. I am also grateful to C. Eastman and A. Perlis, who were kind enough to read early drafts of this paper and to provide many helpful comments.

REFERENCES

1 D L PARNAS J A DARRINGER
SODAS and a methodology for system design
Proc F J C C 1967

2 F W ZURCHER B RANDELL
Multi-level modeling—A methodology for computer system design
Proc International Federation of Information Processing Societies 1968

3 D L PARNAS
Sequential process: A fuzzy concept
Unpublished memo available on request from the author

4 O J DAHL K NYGAARD
SIMULA—An ALGOL based simulation language
C A C M 670–678 September 1966

5 E W DIJKSTRA
Cooperating sequential processes
Technological University, Eindhoven, Netherlands

6 D L PARNAS
Sequential equivalents of parallel processes
Carnegie Institute of Technology Pittsburgh Pa
February 1967

Calculating and plotting equipotential lines for objects with cylindrical geometry

by WILLIAM W. SHRADER

Raytheon Company
Wayland, Massachusetts

INTRODUCTION

A computer program has been prepared that calculates the position of equipotential lines in cylindrical geometry for a multidielectric and multiconductor environment using a modified extrapolated Liebmann method. The output consists of printed voltages on evenly spaced grid coordinates, and an automatic drafting machine drawing of the dielectric and conductor boundaries and the equipotentials. The program has been used to design Corona-free high voltage bushings and terminals, and has been applied to other problems where knowledge of the electrostatic field was desired. The program is an example of how the advent of large, high-speed digital computers has made possible easy solutions of previously intractable engineering problems. Running time on the Univac 1108 computer is typically one to two minutes.

The following are previous techniques used in an attempt to solve this type of problem.

1. The cumbersome tilted electrolytic tank,[1,2]
2. Resistance paper that ignores the cylindrical geometry,[3,4]
3. The tapered resistance analog that represents the cylindrical geometry properly but cannot readily accommodate different dielectric constants,[5]
4. The inflexible approximate analytical methods.[6]

Fortunately the author prepared and used this program before he read Reference 9 which states that this approach will not work for dielectric ratios of 10:1. The author, an engineer, has used the computer as an engineering design tool. The program may not be optimum in a pure mathematical sense, but it does give the required engineering answers quickly and easily.

The method of solution is iterative relaxation using a modified extrapolated Liebmann method[7] The results appear similar to those obtained in Reference 8. Briefly, the problem to be solved is specified in terms of the boundaries of the materials in the region of interest, their dielectric constants, and the voltages, where known, at the boundaries of the region of interest. The computer then lays out a resistance analog matrix of resistors. The value of each resistor is inversely proportional to both the dielectric constant and the distance from the axis of symmetry of the cylindrical geometry. Typical matrix sizes are 48×80, but matrices as large as 100×100 can be accommodated so that adequate accuracy can be obtained without the need to compensate for boundaries that do not fall exactly on a grid point.[5] The computer then solves for the voltage at each point of the matrix in terms of the adjacent four points. This is an iterative procedure. After the computer goes through the matrix several hundred times, the finish criterion is met (the voltage at each point changes between subsequent calculations by less than one part in 10^5), and the resulting voltage matrix is printed. The computer also provides punched cards, if desired, which are the input to an automatic drafting machine for drawing the material boundaries and the equipotential lines.

Sample problem

Following is a description of the evolution of the design of a high voltage feed-through bushing. Figure 1 shows the cross section of the geometry of the problem. The 41kv high voltage is to be fed through air from one tank of oil to another. The pair of bushings is ceramic, and the diameter of the metal center conductor has been enlarged in an attempt to reduce the voltage stress in the air adjacent to the center conductor.

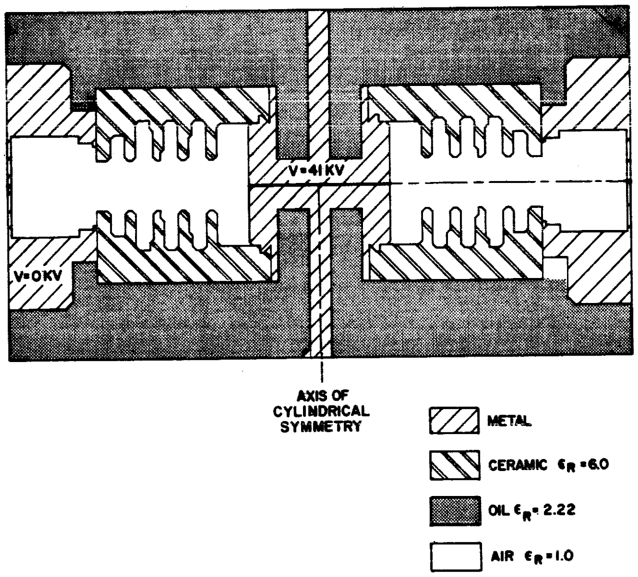


Figure 1—Geometry of high voltage feedthrough

Because of the symmetry of the problem, a solution for the electrostatic field of any quadrant is sufficient. The quadrant analyzed is the lower righthand quadrant as indicated by the dashed line.

Figure 2 illustrates the drafting machine output of the electrostatic field. Excessive bunching of the equipotential lines occurs at the interface between air, ceramic, and center conductor. The stress in the air at this point is 50kv/inch as determined from the computer matrix printout. It was then decided to provide an electrostatic shield for the critical point by undercutting the ceramic and metalizing the undercut surface. The result of providing this shielding is seen in Figure 3. The stress in the air at the junction of the center conductor, ceramic, and air is now 25kv/inch. The stress in the air pocket between the first two ceramic fingers is also about 25kv/inch. There is a bunching of the lines at the intersection of the shield, the ceramic, and the oil, but because this bunching occurs in the oil, it creates no problem.

In another application of this same bushing, a high voltage modulator deck will be located in the oil close to the bushing. Figure 4 shows the analysis of this situation. The maximum voltage stress in air now occurs between the third and fourth ceramic fingers, but it is no greater than the maximum stress in air shown in Figure 3.

The above problem was solved on a matrix grid of 88 by 48 points. The ceramic fingers were not identical in the calculation because the actual ceramic fingers, when traced onto graph paper, did not fall on the grid in the same relative position. The closest matrix coordinates were used for each finger. Examination of the problem on a finer grid has shown that the approximations involved in using the 88 by 48 grid have not compromised the results. There is an edge effect that may occur where the equipotentials meet the boundaries. This effect, particularly noticeable in Figure 4 on the right-hand edge, is a local effect caused by the computer method of estimating the Neumann boundary voltages and should be ignored.

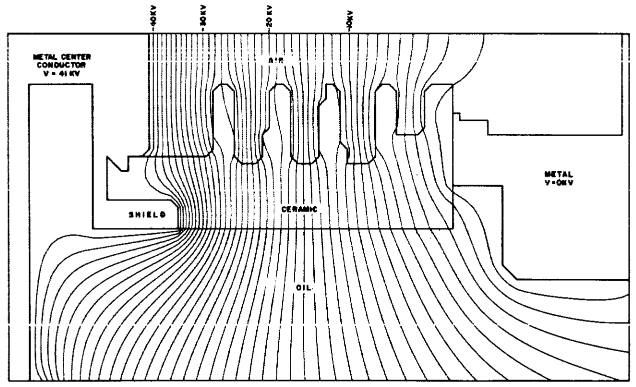


Figure 3—Electrostatic field with modified ceramic bushing

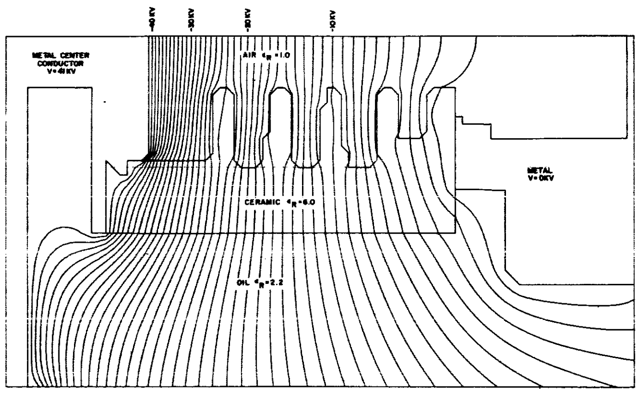


Figure 2—Electrostatic field with original ceramic bushing

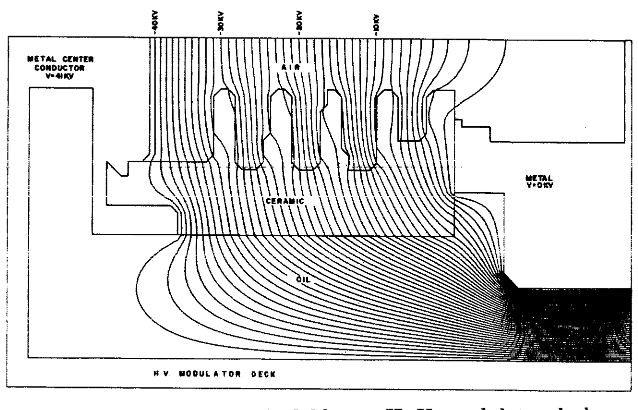


Figure 4—Electrostatic field near H. V. modulator deck

The program

Two concepts are borrowed from techniques used for manual relaxation.[7] One concept is that a solution of the problem obtained on a coarse mesh can be used to estimate the potentials for solution on a finer mesh. The second concept is that with some boundary voltages unknown (Neumann boundaries) a solution of the problem in a region larger than specified can yield accurate estimates of the voltages at the periphery of the region of interest. For the above reasons, the problem is solved in three successive steps where the dimensions of the matrix are increased by a factor of two for each step, and the dimensions of the problem are increased by a factor of four for each step. The final step is the solution of the problem as initially specified.

The problem shown in Figure 3 will be used as an example. The actual shape of the pieces are traced on graph paper; then the nearest coordinates on the grid are selected for describing each piece as shown in Figure 5. The tapered ceramic fingers were identical before the nearest grid points were selected to describe them. When the voltages on the boundary are known, they are specified as fixed voltages. This is true for the 410 volts (representing 41 kv) on the center conductor and 0 volts on the metal on the right-hand edge. The Neumann boundary, along most of the bottom edge and part of the right edge, is specified as an even gradient from 410 to 0 volts on the bottom edge, and 0 on the right edge. Specification of the Neumann boundary is not critical because this value will only be used for the first, coarse solution of the problem.

The program reduces the mesh size by a factor of 4 and the coordinates of each piece by a factor of 16 (unless the coordinate lies on a boundary; then the coordinate is reduced by 4). This results in the problem appearing as in Figure 6. This problem is solved, and the voltages within the dashed line are used as initial voltage estimates for the next, more refined, solution. The voltages on the dashed line are the Neumann boundary voltages for the next solution. The second solution uses a mesh one-half the size of the final mesh, and the coordinates of each piece are reduced by a factor of 4 as illustrated in Figure 7. The third solution is the final, full size solution. The initial voltage estimates obtained from the first and second solutions generally vary only a few percent from the final voltages of the subsequent solution.

To solve the problem, the program lays out a matrix of resistors which connect each point on the grid with the adjacent four points. The voltage at each point is successively calculated. The calculation at each point is based on the voltage at the four adjacent points (refer to Figure 8). Values of E_1 through E_5 are stored in the memory from previous calculations. G_1 through G_4

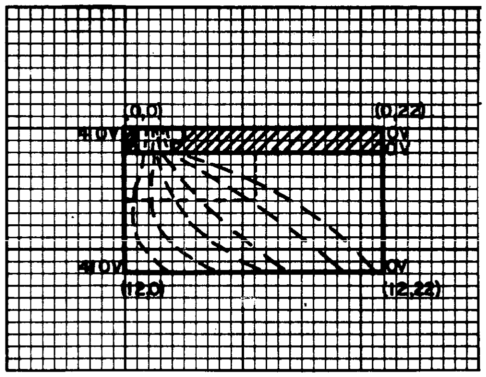


Figure 6—First solution

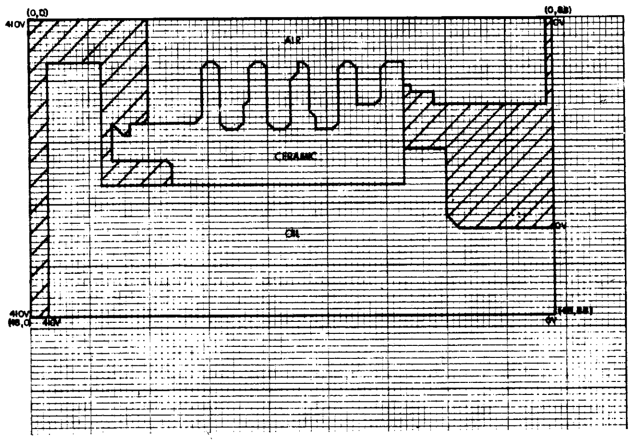


Figure 5—Problem as specified

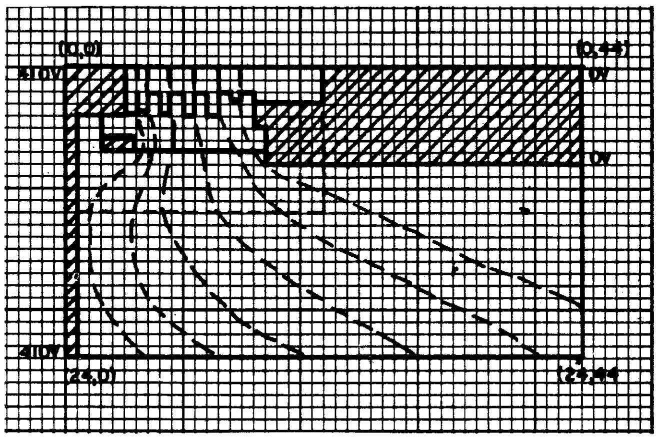


Figure 7—Second solution

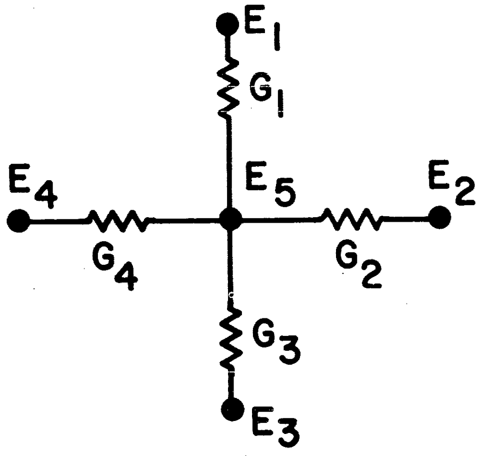


Figure 8—Resistor matrix

represent the conductances of resistors tying the adjacent grid points together. The new value of E_5 would be:

$$\left(\frac{E_1 G_1 + E_2 G_2 + E_3 G_3 + E_4 G_4}{G_1 + G_2 + G_3 + G_4} - E_5\right) \text{RELF} + E_5 \quad (1)$$

RELF is the relaxation factor[7] (also called the acceleration factor or convergence factor). RELF should lie between one and two; the speed of convergence of the problem depends on the value of RELF chosen. This program uses:

$$\text{RELF} = 2\left(1 - 0.8\pi\sqrt{\frac{1}{p^2} + \frac{1}{q^2}}\right) \quad (2)$$

where the matrix has (p + 1) × (q + 1) points. For example, a matrix 48 × 88 uses RELF = 1.88. Originally the equation is reference 7 page 273 was used; this equation is identical to the above equation with the 0.8 replaced by unity. For the few problems for which a comparison has been made, the above equation leads to faster convergence than when the 0.8 is replaced by either 0.6 or unity.

The criterion that the iteration is finished is that no point changes on successive iterations through the matrix by more than one part in 10^5 of the maximum voltage difference in the matrix. Thus in the example given, no voltage changes by more than 0.0041 volt on the last iteration. This does not mean that the final voltages are that precise, for spot checks have shown the absolute accuracy in places on the grid may be only one part in 10^3, but this is believed sufficiently accurate for most engineering applications. The two preliminary calculations (Figures 6 and 7) use one part in 10^3 and one part in 10^4 respectively as the finish criterion.

The first solution (Figure 6) of the sample problem converged in 21 iterations; the second solution (Figure 7) converged in 43 iterations; and the third solution (Figure 5) converged in 89 iterations.

The practical problems examined so far have had the relative dielectric constants not vary by more than a factor of 10. Metal pieces, that in the resistance analog are highly conductive, have always been attached to a boundary where the voltage is given. To determine if convergence would occur with a greater ratio of dielectric constants, the sample problem was rerun with a piece of material with a dielectric constant of 25 inserted in the air. This problem converged in 26 seconds after 125 iterations for the third solution with the results shown in Figure 9.

Detailed description

The problem is laid out on graph paper where either the horizontal axis or the vertical axis is the center of the cylindrical geometry. It is assumed that the voltages on the vertical axis are known (such as in the sample problem where the vertical axis is the center conductor) and the voltages on the horizontal axis are unknown. The matrix size is determined, and the boundaries of the dielectric pieces and the conductors are placed on the grid. All coordinates are specified as row and column grid coordinates starting at 0, 0 in the upper left-hand corner. Thus a matrix specified as 48 by 88 will actually have 49 by 89 points. This choice of input coordinates simplifies both specifying the problem and scaling between successive solutions, but it complicates handling the matrices stored in the computer which must be subscripted, for example, from 1 to 49 and 1 to 89.

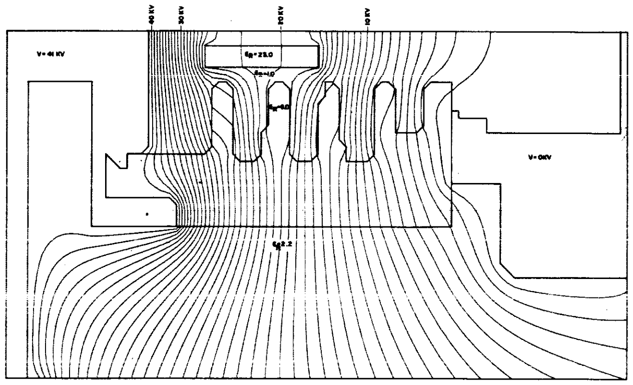


Figure 9—Example showing convergence with dielectric ratios of 25:1

The program reads the input data in the following sequence.

1. The matrix size (48 $\times$ 88) and a code number to indicate which axis is the axis of symmetry.
2. The coordinates and voltages of each point specifying the Neumann boundaries. The program interpolates between the specified points for the first calculation of the problem.
3. The coordinates and voltages of the points specifying the fixed voltages. The computer interpolates between the specified fixed voltage points and sets the voltages on the boundaries to these values before each solution of the problem.
4. The coordinates and relative dielectric constants of each piece of material. Metal is represented by specifying a very high dielectric constant (100,000.).
5. The voltage increment for drawing the equipotential lines, if desired.

Three matrices are stored in the computer. The first contains the voltages at each point (dimensioned as E(101, 101)); the second contains the conductances of the resistors to the left of each point and below each point (dimensioned as G(101, 101, 2)); the third contains the sum of the conductances of the resistors meeting at each point (dimensioned as SUMG(101, 101)). Although the space in computer memory is reserved for matrices of 100 $\times$ 100, not all of this space is used for smaller matrices.

The program examines the fixed input voltages, and calculates 1/1000 of the difference between the maximum and minimum. This is used as the finish criterion for the first solution of the problem. This value is multiplied by 0.1 and 0.01 for the next two solutions.

The size of the initial matrix is calculated to be the specified size divided by four (12 $\times$ 22). Next, the coordinates of the Neumann boundary voltages are scaled to match the initial matrix size, and the voltages are calculated for each point in the voltage matrix on the Neumann boundary.

The scale factors for scaling the voltage coordinates and the material coordinates are set for the first time through the problem; the main portion of the program is now entered. This main portion of the program is re-entered for the second and third time through the problem with only a change being made to the scale factors.

The scaling of coordinates, whether for Neumann boundary voltages, fixed boundary voltages, or the conductor or dielectric pieces, is accomplished as follows. If the row (or column) coordinate equals the height (or width) of the matrix, it is reduced by a factor of 4; otherwise it is reduced by a factor of 16. Thus all points specified as being on the matrix boundary remain on the boundary. For example, the coordinates of the Neumann boundary voltage which was specified as 0 volts from point (34, 88) to point (48, 88) are scaled to (2, 22) and (12, 22). (The coordinates are rounded to the nearest integer.) The scale factors of 4 and 16 above are changed to 2 and 4 for the second solution and 1 and 1 for the final solution.

The main portion of the program:

1. Scales all the coordinates
2. Sets the fixed voltages on the matrix boundaries
3. Calculates the resistor matrix
4. Calculates the matrix of the sum of the conductances
5. Calculates the relaxation factor
6. Iterates the calculations of the voltage matrix until the finish criterion is met.

After the first or second time through, the finish criterion is multiplied by 0.1, the two scale factors are divided by two and four respectively, the voltage matrix is interpolated for estimating the voltage matrix for the next solution, and the main portion of the program is reentered. After the third time through, the resistance matrix is printed, and the voltage matrix is printed. Finally if an input value is given for spacing of the equipotential lines, the punched card output is prepared for the automatic drafting machine.

Of the above steps, only two need further comment: the calculation of the resistance matrix, and the calculation of the equipotential lines.

The resistance matrix is laid out in the sequence the materials were specified. The materials should be specified in order of increasing dielectric constants, with the conductors specified last. Thus the boundary of two pieces will represent the conductivity (or the conductivity analog of the dielectric) of the piece on the boundary with greatest conductivity. The coordinates of the pieces are specified on the mesh points. They may be spaced any distance horizontally or vertically, but may only be spaced one mesh square apart diagonally. The points are specified in a clockwise direction around the periphery of each piece.

The program examines the points in sequence. If a point lies anywhere to the right or directly below the previous point, resistors are inserted in the matrix. If a point lies to the left or is identical to the previous point (which may happen due to the scaling process), it is ignored. When the resistors are inserted, they are inserted from the point vertically downward until a boundary of the piece is reached. The boundary is determined by examining the remainder of the points specifying the piece, looking for the nearest-below, right-to-left crossing of the boundary of the piece. Debugging the logic of where resistors should be inserted for the many possible shapes of materials required about 95 percent of the debugging effort of the program.

As mentioned earlier, the conductance of each resistor is stored in the resistance matrix. If the axis of symmetry is the vertical axis, conductance of the resistor to the left of the point at row I, column J, is:

$$G(I, J, 1) = (2J - 1)\,D \qquad (3)$$

where D is the dielectric constant. The conductance of the resistor below point (I, J) is:

$$G(I, J, 2) = 2\,JD \qquad (4)$$

If the axis of symmetry is the horizontal axis, the value of each conductance is:

$$G(I, J, 1) = 2\,ID$$

except where $I = 0$, where

$$G(0, J, 1) = D/4 \qquad (5)$$

and

$$G(I, J, 2) = 2(I + 1)\,D$$

The I and J in the above equations are based on the mesh where the coordinates of the upper left corner are specified as (0,0). The equation for G(0, J, 1) for the horizontal axis of symmetry is commensurate with the computer method of computing the E(0, J)'s, which assumes a mirror E(−1, J) equal to E(1, J), and a mirror G(−1, J, 2) equal to G(0, J, 2).

After the resistor matrix has been calculated, it is examined for high values of conductances. All values of conductance above 9999 are set to 10^{10}. This ensures that the conductors are not influenced by the voltages in the surrounding dielectrics. This also places the limitation on the program that all conductors must be attached to a boundary. (Before printing the resistance matrix, all conductances of 10^{10} are set to 99999 so that integers may be used in the printout.)

Each equipotential line is calculated in the following manner. Each row of the voltage matrix is searched for the specified voltage. Linear interpolation is used between grid points higher and lower than the specified voltage. Note that a given equipotential may cross a row more than once. After all crossings of each row are determined, all columns are searched for the specified voltage. This completes the list of all points where the equipotential crosses the lines of the mesh. These points are then searched to find one on the mesh boundary. Starting with the point on the boundary, all the other points in the list are searched to find the closest one. The process is repeated until all points are ordered in sequence.

ACKNOWLEDGMENTS

I am indebted to Tom Weil for suggesting the problem and providing the necessary background and references. Ed Maybury has provided excellent continuing guidance in the use of the Univac 1108.

REFERENCES

1 W J KARPLUS
Analog simulation
133 McGraw-Hill Book Co New York 1968

2 P A EINSTEIN
Factors limiting the accuracy of the electrolytic plotting tank
Brit J Appl Phys Vol 2 49–55 February 1951

3 J S BONNESEN
A corona free high voltage feedthrough terminal design for electronic applications
Proc IEEE Electronic Components Conference 1965

4 J A ROSS
Submersible, pre-cast cable terminators for splicing and terminating at distribution voltages
Permali Inc Mount Pleasant Pennsylvania

5 J R HECHTEL J S SEEGER
Accuracy and limitations of the resistor network used for solving Laplace's and Poission's equations
Proc IRE Vol 49 No 5 933–940 May 1961

6 H MCL RYAN C W WOLLEY
Sparking voltage of a conductor passing through an earthed plate
Proc IEE Vol 114 No 1 172–178 January 1967

7 K J BINNS P J LAWRENSON
Analysis and computation of electric and magnetic field problems
265–276 The Macmillan Company New York 1963

8 J A SEEGER
Solution of Laplace's equation in a multidielectric region
Proc IEEE letters Vol 56 No 8 1393–1394 August 1968

9 R H GALLOWAY H MCL RYAN M F SCOTT
Calculation of electric fields by digital computer
Proc IEE Vol 114 No 6 824–829 June 1967

10 D W PEACEMAN H H RACHFORD JR
The numerical solution of parabolic and elliptic differential equations
J Soc Indust Appl Math Vol 3 No 1 March 1955

A modular system for reactor calculations*

by L. JUST, A. KENNEDY, P. WALKER, A. RAGO**
and G. LEAF

Argonne National Laboratory
Argonne, Illinois

INTRODUCTION

The ARC (Argonne Reactor Computation) System has been developed to facilitate the studies required for fast-reactor design. The areas of physics, safety, fuel utilization and core-design had initial priority. Other general engineering capabilities will be added later.

Reactors have traditionally been designed by means of stand-alone codes. However, it is rarely the case that a single code will suffice for an entire calculation much less for an entire reactor design. Usually, the solution of one problem becomes the input to another problem, but not without the intervention of conversion programs. This procedure is followed until the complete solution is effected.

It was inevitable that the reactor industry would sponsor the development of systems to automate the design process. The first effort in this direction was undertaken by the Knolls Atomic Power Laboratory with the NOVA project in 1964. The ARC project at Argonne National Laboratory was begun in 1965. Still later, the JOSHUA project was begun in 1968 at the Savannah River Laboratory.

The ARC system, when completed, will encompass the field of reactor computations and automate what was once a series of interrelated programs. A glossary has been constructed that defines and formats all of the types of data that can be expected for reactor calculations. In addition, a library of programs has been built that will perform the mathematical and data handling algorithms that occur in reactor computations.

A "system" capability has been provided that allows the library and data to be used in a flexible, coordinated manner. Reactor calculations can be accomplished by designing a director program that executes modules from the library in the proper sequence to operate on and produce data.

Although such a system can be specified independently of a choice of computer, the implementation is not computer independent; it depends upon a choice of machine and the manufacturer's software. ARC is currently running on an IBM System/360 50–75 combination using ASP. The system runs as an ordinary job under OS/360 and it will run on any configuration which uses OS/360. Practically speaking, ARC needs a large amount of core and auxiliary storage, and a fast CPU. These restrictions are caused by the nature of reactor calculations rather than the nature of ARC; reactor calculations involve a lot of computation and require extensive data manipulation and storage capabilities.

Design features

Original specifications

The original specifications for ARC required the following features:

1. Modular approach. Any major computational program consists of a collection of computational units, certain of which are common to many computations. A modular approach facilitates the construction of such programs. In addition, the modular approach provides a maximum amount of core for each module provided the system storage requirements remain small.
2. Common data base or data pool. In a modular approach the data associated with each module must be potentially available to any other module in the system. This implies the existence of a

*Work performed under the auspices of the U.S. Atomic Energy Commission.

**Present address: IBM Scientific Center, Palo, Alto, California.

data pool which is responsible for intermodule data management.
3. All coding in FORTRAN IV. If all coding is done in high-level languages, then program interchange is greatly facilitated.
4. Use of manufacturer's software whenever possible.
5. Ease of use. This is probably part of the specifications for all systems and is an area where much time can be spent even after a system becomes operational.

Original implementation

Surprisingly, a simple initial approach satisfied all of the criteria except 5. This approach was a huge overlay program with system functions in the root segment, paths in region one and computational modules in region two.

The overlay system shown in Figure 1 was in production before it collapsed of its own weight. Two flaws dictated the approach described in this paper:

1. Overlays allow a limited number of external symbols, and duplicate names are forbidden.
2. The system had to be reconstructed whenever *any* change occurred anywhere in the system.

A small amount of assembly language coding turned the overlay system into a dynamic loading system. The computational modules of the overlay were assimilated into the present system with few changes.

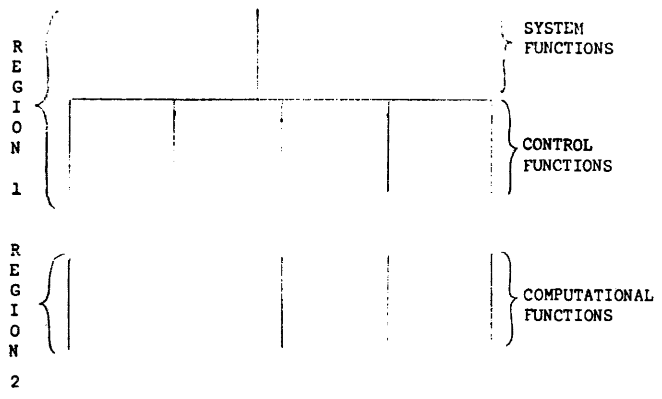


Figure 1

The dynamic system

The main components of ARC

1. A library of programs (OS/360 load modules).
2. ARC system subroutines that are useful to most modules of the library.
3. A data set glossary that defines the data sets.

In addition, the ARC system is predicated on the use of OS/360 with data in the form of OS/360 data sets.

Modules in ARC

ARC contains three kinds of modules, computational, control, and system.

1. A computational module is defined in terms of its input, output, and the algorithms that manipulate the input to produce the output. Input to computational modules is in the form of data sets and parameters lists; output is in the form of data sets and printed output. All computational modules are written in FORTRAN IV and are loaded and executed by control modules.
2. A control module is a FORTRAN IV program that can be executed as the main program of ARC. A control module or *path* is directive in nature; it causes computational modules to be executed in the proper sequence to effect a series of calculations. A path performs the following functions:

   a. Initialization of core to provide an environment for the computational modules.

      i. Permanently loads resident system modules.
      ii. Initializes tables.
      iii. Processes card input for later use.

   b. Directs the computation.
   c. Calls for subsets of the input data as they are required.

   Paths are classified as *standard* if they exist in the library at the start of a run or *nonstandard* if they are compiled at run-time. Any path is allowed to use another path as it would any computational module. This allows a new calculation to use an existing calculation if it performs a useful subcalculation.

   In order to initialize the system, a path must contain two initialized arrays. The first array contains the names of all of the data-sets used by the path and by any module called by the path. The placement of the name in the array determines the data set reference number associated with the name. The second array lists the names of blocks of input data that are required to perform the calculation. Input data is divided into blocks with cards of the form

BLOCK = $NAME_n$

where $NAME_n$ is an 8-character identifier. When a path performs

CALL DATA(NAME, N),

a table that describes the input data set is searched for the occurrence of the block NAME. If a block by that name that has not been processed exists, it is formed into data sets. Block names can be duplicated, and repeated calls of the same name process the next block by that name; when all blocks with that name have been processed, that fact is reflected in N. This facility is useful in repeated runs of a path. The absence of a block can terminate the run or cause another logical decision to be made.

3. ARC *system modules* perform functions that are required by paths and computational modules. They are of two types:
   a. Resident. These are permanently loaded into core and executed by the first path to gain control. Resident system modules perform the functions of system initialization, input data processing, data management, and internal table maintenance.
   b. Transient. These modules are executed by certain of the resident system modules. Their primary function is to process input data. One module is used during initialization to convert input data into a data set and to describe the properties of that data set in tables held in core. Another module is used when a path requires a subset of the input data.

System subroutines

The second main component of ARC is a collection of *system subroutines* which form an essential part of most of the modules in the system. These subroutines allow the FORTRAN program to invoke certain of the OS/360 macros, provide an interface to the resident I/O module, and communicate with the system. A group of system subroutines, collectively called POINTR, provides a variable dimensioning capability and is included in the modules that require this feature.

The system subroutines LINK and LOAD invoke the MACROS with the same names.

1. LINK is an assembly-language subprogram callable from a FORTRAN subprogram via the statement

   CALL LINK (name, additional parameters)

   where "name" is either an 8-character alphameric quantity or a "REAL*8" variable containing such a quantity. The quantity must be the name of an ARC module.

   The subprogram LINK causes the OS/360 "LINK" macro-instruction to be invoked and results in control being transferred to the entry point of the module "name." If "name" is not currently in core, the "LINK" macro loads "name" into unoccupied core before giving it control. Any additional parameters in the above call to LINK are passed to "name." When control exits from "name," it returns to the calling subprogram at the statement immediately following the call to LINK.

   LINK may be used to pass control from one program module to another in exactly the same fashion as the FORTRAN "CALL" statement is used to pass control from one subprogram to another within the same module.
2. LOAD is an assembly-language subprogram callable from a FORTRAN subprogram via the statement

   CALL LOAD(name)

   where single parameter is either an 8-character alphameric quantity or a "REAL*8" variable containing such a quantity. The quantity must be the name of an ARC program module.

   The subprogram LOAD causes the OS/360 "LOAD" macro-instruction to be invoked and results in the loading of the module "name" into unoccupied core. If the module "name" is already resident in core, then either (i) no action is taken (if the module was declared reusable), or (ii) another copy of "name" is loaded (if the module is not reusable).
3. Modules locate data by using the system subroutine DSRN. DSRN is called from a FORTRAN subprogram via the statement

   CALL DSRN (name, N1, N2)

   where "name" and N2 are input parameters, and N1 is an output parameter.

   "Name" must be either an 8-character alphameric quantity or a "REAL*8" variable containing such a quantity; that is, one of the glossary names in the array DSNAME as

initialized in the path control module of the path being executed.
The function of DSRN is to return to the calling program in N1 a data set reference number corresponding to the data module "name." In this way data set reference numbers are global to the system instead of local to the modules. The subroutine DSRN simply LINKS to a resident system module that performs the table search.

4. Data management within fast memory is accomplished by the subroutines of POINTR. Each subroutine of POINTR performs a specific data management function and as such provides a "pseudo-instruction" in a set of data management instructions. Thus, the FORTRAN language has been extended to allow data packing, storage allocation, and storage freeing in a manner similar to that available in the PL/I language.
Control of data in fast memory remains under the control of the module programmer, who provides POINTR with a large "container" array into which data arrays will be placed. POINTR contains tables which keep track of arrays being stored in the container, thus relieving the programmer of this burdensome chore. Recorded in the tables are the array name (usually identical with the FORTRAN variable name), length, starting location, and FORTRAN variable type. Subsequent retrieval of arrays is by array name.
5. Management of external data within a module is through the standard FORTRAN I/O routines and the data set is the organizational unit. This obvious approach simplified the initial programming but created a rather rigid data structure. A large amount of effort went into the construction of a glossary that describes all of the data sets that have been written. The glossary is open-ended and at all times contains the definitions and description of all data sets which are used or created in ARC. In addition, the first record of most data sets lists the contents and important parameters.

This discussion of the components of ARC shows that OS/360 is a foundation of the system. Through the implementation, a guiding philosophy has been "peaceful coexistence with OS/360."

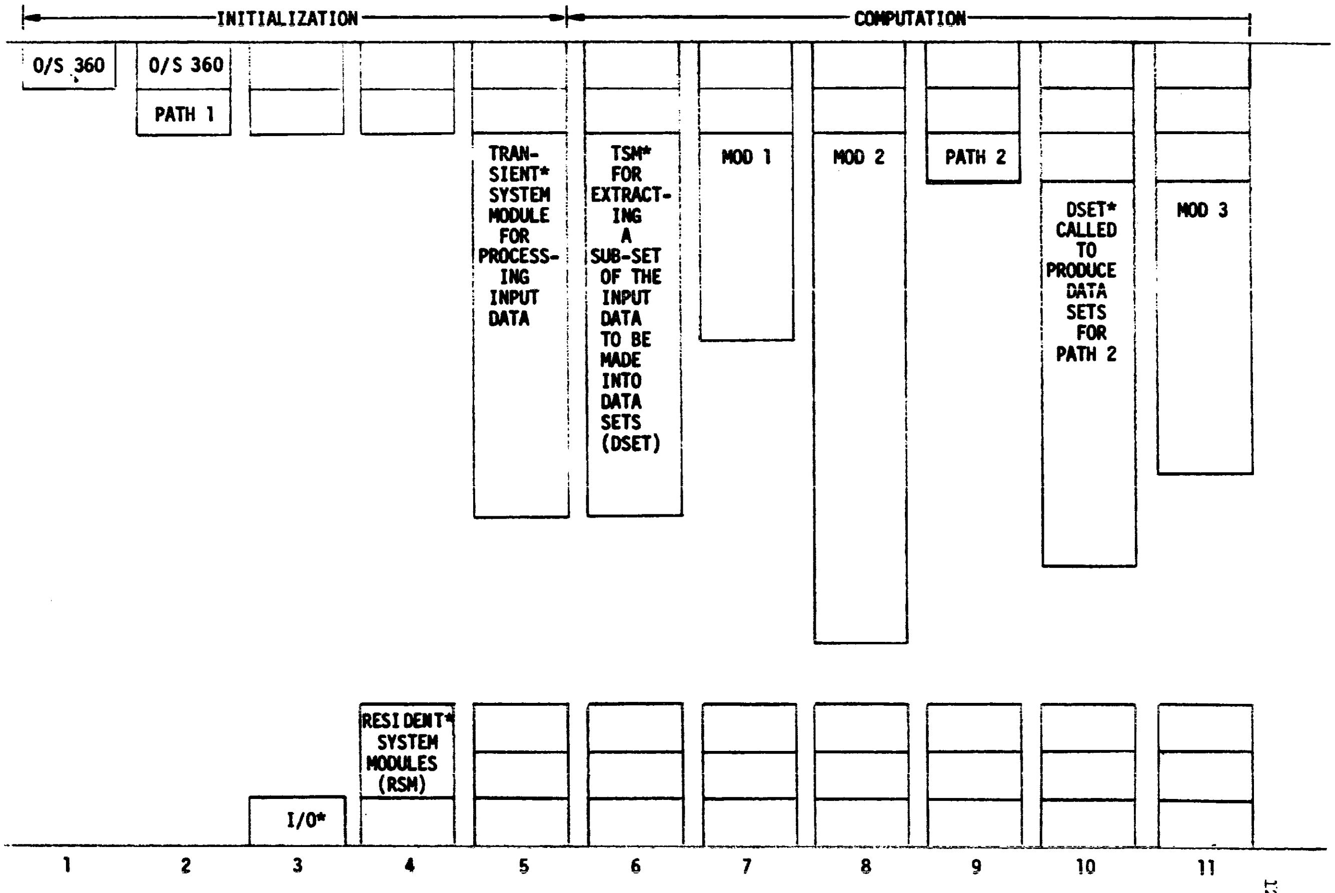

INITIALIZATION
COMPUTATION
O/S 360
O/S 360
PATH 1
TRANSIENT* SYSTEM MODULE FOR PROCESSING INPUT DATA
TSM* FOR EXTRACTING A SUB-SET OF THE INPUT DATA TO BE MADE INTO DATA SETS (DSET)
MOD 1
MOD 2
PATH 2
DSET* CALLED TO PRODUCE DATA SETS FOR PATH 2
MOD 3
RESIDENT* SYSTEM MODULES (RSM)
I/O*
1
2
3
4
5
6
7
8
9
10
11

Figure 2

A sample program

The following statements show the coding that is necessary to compile and execute a nonstandard path (PATH1). Execution of PATH1 causes the events depicted in Figure 2 to take place. Figure 2 is a dynamic picture of core with time increasing from left to right. ARC system modules are marked with an asterisk.

```
        The following statements show the coding that is necessary to
             compile and execute a nonstandard path (PATH1).

//NSP JØB
//JØBLIB DD DSNAME=MØDULES,DISP=(ØLD,PASS)
//STEP1 EXEC FTHCLG
//FTH.SYSIN DD *
C            DD
C     PATH1
      REAL*8 DA(240)/'A.A','A.B','XYZ','$'/                          STMT1
      REAL*8 DS(10)/'BLOCK1','BLOCK2'/                               STMT2
C
      CALL SYSTEM(DA)                                                STMT3
      CALL DATA(DS(1))                                               STMT4
      CALL LINK('MOD1     ')                                         STMT5
      CALL LINK('MOD2     ')                                         STMT6
      CALL LINK('PATH2    ')                                         STMT7
C
C
      RETURN                                                         STMT8
      END                                                            STMT9
/*
//EDT.LIB DD DSNAME=MØDULES,DISP=(ØLD,PASS)                          STMT10
//EDT.SYSIN DD *                                                     STMT11
 INCLUDE LIB(SYSTEM)                                                 STMT12
/*
//GØ.FT09F001 DD DISP=NEW,UNIT=DISK,SPACE=(TRK,(10,10))
//GØ.FT11F001 DD DSNAME=A.A ........
//GØ.FT12F001 DD DSNAME=A.B ........
//GØ.FT13F001 DD DSNAME=XYZ ........
//GO.SYSIN DD *
BLOCK=BLOCK1
DATASET=A.A
 {Data for A.A}
DATASET=A.B
 {Data for A.B}
BLØCK=BLØCK2
MØDIFY=A.A
 {Cards to modify data set A.A}
/*
```

The FORTRAN IV program PATH1 is compiled during time period 1 (TP1). At TP2 the path is in core and the I/O initialization of FORTRAN causes the ARC I/O module to be LOADed into core. TP4 and TP5 are related to the system initialization and make use of STMT1, STMT2, and STMT3 of PATH1. STMT3 actually causes system initialization utilizing the information in STMT1. Three data sets are expected to be used (A.A, A.B, XYZ) and input data are divided into two blocks (BLOCK1 and BLOCK2).

STMT4 corresponds to TP6; it causes BLOCK1 to be processed into data sets A.A and A.B. STMT7 is responsible for TP9, TP10, TP11. PATH2 processes BLOCK2 into a modification of data set A.A. In STMT12, a module named SYSTEM is being combined with PATH1. SYSTEM contains all of the ARC system subroutines that a path requires.

This simple example shows that the internal workings of the system are simple and flexible. Since a path is a FORTRAN program, complex logical structures pose no problems; they are composed of familiar statements. The computational modules are also composed of FORTRAN statements and the only restriction is that a computational module never tries to initialize the system.

Current capability

Our program library presently contains 40 modules. Twenty-two of these are computational modules, twelve are control modules and six are system modules. In addition there are nine system subroutines that are part of each computational and control module. These subroutines interface with the system modules.

The average computational module contains about 2500 statements. About half of the computational modules are overlay programs and subroutine standardization is utilized as much as possible.

Critique and future plans

Problem areas

ARC is an ambitious project pursued by a small number of people. As such the project has been subjected to compromises dictated by limitations in manpower and time. Early in the project a decision was made to expend the greatest effort on the computational aspects of the system and to rely as much as possible on OS/360. Consequently some of the desirable features of the originally specified system were deferred or compromised. A brief criticism of the present system will be enlightening.

Any large, modular scientific programming system is dominated by the problems of data management. These problems are in two areas:

1. management of program modules, i.e., the ability to link computational modules in a complex manner;
2. management of computed data, the ability to store and retrieve data.

Program management

While adequate linkage facilities are available to the module writer, job executions are not convenient for the user. The Data Definition statements of OS/360 are the external data language for this system. The quantity and complexity of these DD statements cause many errors to be made. Thus, the reactor designer may have little difficulty in designing and writing a new path, but have great difficulty making it work. In practice, the programmer must be quite adept to some of the most sophisticated uses of OS/360.

Data management

The method of computed data management within the ARC system is also not satisfactory for a third-generation system, both from the viewpoint of the reactor designer and the program developer. In any large system there is always an inherent conflict between the structure of the language of the user and the internal structure of data necessary for efficient operation of the system. In ARC we have compromised. The Data Definition statement of OS/360 is not oriented to the reactor designer. The rigidly structured external files used in ARC cause two classes of problems.

1. They are not conducive to efficient manipulation other than for the specific use that dictated their initial structure. Future arbitrary use and manipulations of these files or, more importantly, arbitrary combinations of parts of these files must necessarily be time-consuming. If, for example, subsequent references require that the file structure should be inverted, the ordering is in conflict for efficient operation; etc.
2. Modularity of program development is greatly hampered by the rigidly structured files because redefinition of the file structure in files produced by any module must be reflected in all modules referencing that file. This modularity would be enhanced if data references were instead oriented to FORTRAN variables. Further complexity is introduced since the module developer must concern himself not only with data of interest to his module but all quantities in each external file necessary for him to reference.

Proposals for improvement

These problems have generated specifications for two subsystems:

1. A generalized data management package that would allow modules to reference all data by FORTRAN variable names irrespective of core storage requirements. This package would have taken on the responsibility of all core manage-

ment and external file manipulation by means of high frequency, priority usage algorithms. Thus it would be transparent to the module writer if variable names that were referenced in the module required external file manipulation or not; i.e., the system would provide him with an effective infinite memory. This proposal addressed itself to the total problem of management of computed data as well as enhancing modularity of program development. This ambitious project was not undertaken due to lack of resources.

2. A system that addresses itself to the problems of program management and external file management. This system is comprised of a program maintenance package, a datapool maintenance package, and a job control language translator. This too is an ambitious project that would have general usage outside of the ARC system.

Current plans

Our current plans are evolutionary rather than revolutionary. Three aspects must be considered:

1. Internal algorithms.
2. Data management.
3. External language.

The internal algorithms are constantly reviewed and are improved wherever possible. Improvements in algorithms should not negate work that has been done in other areas.

Data management will be augmented; most of the data sets that have been defined will remain, but alternate facilities will be provided. In particular, we expect to construct a variable-oriented data pool. When this is completed, some of the data sets will be separated into many members that can be accessed directly. In particular, certain important variables will be referenced by name throughout the system.

An external language is required to ease the burden of running a computation that is in production status. The deck of Job Control Language that is required to run a job is large. While it is true that a user can accomplish all of the external data set manipulations that are required, he must specify his needs in JCL, and this is clumsy. Preliminary plans have been made for a simplified language that will produce the JCL required to run any job that is in production status.

This system, while designed for the reactor industry, has generality. With a suitably defined glossary and computational modules, it can be the basis for a computational system for other areas.

REFERENCES

1 B J TOPPEL
Reactor computation development
Reactor Physics Division Annual Report July 1 1964 to June 30 1965 ANL-7110 p 339

2 E D REILLY JR W H TURNER
The automation of reactor design calclations at KAPL
Proceedings of the Conference on the Application of Computing Methods to Reactor Problems May 17-19 1965 ANL-7050 p 251

3 B J TOPPEL
The Argonne reactor computation system
Reactor Physics Annual Report July 1 1965 to June 30 1966 ANL-7210 p 349

4 C N KELBER G JENSEN L JUST B J TOPPEL
The Argonne reactor computation system ARC
Proc of the International Conference on the Utilization of Research Reactors and Reactor Mathematics and Computation p 1428

5 B J TOPPEL
The Argonne reactor computation (ARC) system
Reactor Physics Division Annual Report July 1 1966 to June 30 1967 ANL-7310 p 433-436

6 B J TOPPEL
The Argonne reactor computation system ARC
ANL-7332 1968

7 L C JUST S D SPARCK
The ARC system
AMD Internal Technical Memorandum No 157 1968 unpublished

8 J E SUICH J C JENSEN H C HONECK
JOSHUA—An operating system for reactor physics computations
Proceedings of the Conference on the Effective use of Computers in the Nuclear Industry Knoxville Tennessee April 1969

Performance testing of function subroutines*

by W. J. CODY

Argonne National Laboratory
Argonne, Illinois

INTRODUCTION

The certification of numerical subroutines for an automatic computer is generally a difficult job. In one sense, certification of a subroutine is an in-depth review of its documentation to determine how well the documentation describes the performance and programming of the subroutine. It involves, among other things, performance testing of the program being certified. Of necessity, the details of the performance testing vary from one type of subroutine to another. Indeed, in most cases it is difficult to define a concrete measure of performance for a particular type of routine. For subroutines for computing functions, however, measures of performance are rather easy to define[1,2,3] and much effort has gone into certifications in the past few years.[3,4]

The present paper concerns itself primarily with performance testing of function subroutines and ignores most other aspects of certification. We set forth certain basic principles for such performance tests, examine the rationale behind them, and present in detail a simple yet effective technique for testing function subroutines.

In what follows, we use the term subroutine to mean a subroutine for the evaluation of a real function of a single real variable.

Preliminaries

At the outset we must distinguish between actual testing of performance of a subroutine and judgement of quality based upon the results of the testing. The first process should be a rigorous one resulting in indisputable facts and statistics, while the second is an interpretation of the test results which often depends upon the intended usage of the routine. E.g., for many Monte-Carlo uses, a subroutine that produces only two or three significant figures is adequate, while errors larger than a few units in the least significant figure of a result may be intolerable in precise computations. Clearly, there is an intimate relationship between performance testing and the possibility of judgement of quality. If a testing procedure is designed to determine precise errors any user can determine whether or not the routine is accurate enough for his purposes, but if the tests are limited to verifying gross accuracies the results are useful only to a Monte-Carlo user. Therefore, to be meaningful to the widest class of potential users quality testing should attempt to determine errors as precisely as possible. Of course, there is much more to testing than determining numerical accuracies, but let us concentrate on this aspect first.

We can divide the error made by a subroutine into three parts: transmitted error, analytic truncation error and arithmetic rounding error. The first of these can be described as follows. Let

$$y = f(x)$$

be the function being computed, and assume that our subroutine is infinitely precise, i.e., that given an exact value of x we can compute the corresponding value of y exactly. Then, assuming f(x) is differentiable

$$\frac{dy}{y} = x\frac{f'(x)}{f(x)}\frac{dx}{x}. \tag{2.1}$$

The quantity

$$\delta z \approx dz/z$$

is called the relative error in z and is related to the number of correct significant figures in z. Thus, equation (2.1) establishes the approximate relation

$$\delta y \approx x\frac{f'(x)}{f(x)}\delta x \tag{2.2}$$

* Work performed under the auspices of the U.S. Atomic Energy Commission.

between the relative accuracy of the argument x and the relative accuracy of the corresponding function value y. By the very nature of the function being computed, an inherited error, δx, in the input argument is transmitted by even a perfect subroutine into a corresponding transmitted error, δy, in the function value.

A perfect subroutine is rarely attained. In most cases the computation is based upon the first few steps of an essentially infinite process, such as the evaluation of the first n terms of an infinite expansion. This finite approximation to the infinite process introduces an analytic truncation error, which we denote by T(x). Since a computer does arithmetic on a finite subset of the real numbers, and treats only numbers with a given number of significant figures (bits, bytes, etc.), the computation also involves arithmetic roundoff error which we denote by R(x). Thus, the total error in the computation is

$$\delta y = x\frac{f'(x)}{f(x)}\delta x + T(x) + R(x). \qquad (2.3)$$

The truncation and roundoff error is grouped together below into what we call the generated error.

The black box philosophy of testing

In practice it is rare for a result to be devoid of transmitted error. The fact that computers are basically binary devices while initial data to a program are usually expressed in the incommensurable decimal system is the first major source of inherited error in function arguments. Even if the original data to a program are "clean" binary data, they can quickly become contaminated by roundoff error in the machine. Thus the argument prepared for a function subroutine is seldom exact. Since the subroutine has no control over the inherited error, it cannot be held responsible for the transmitted error. In this sense the subroutine must be considered as a "black box," a separate entity which naively assumes that all arguments supplied are exact and attempts to return a machine number close to the corresponding function value.

The term "black box" has many connotations. There are those who rightfully object to being handed a routine which purports to compute a given function and being asked to accept this claim on faith. The blackness in this type of "black box" is a lack of documentation, *including certification.* We do not claim that a function routine should be a "black box" in this sense. We do say that it should *behave* as a "black box," oblivious to the program around it and aware only of the arguments it receives and the results it returns. Performance testing should treat it as such a "black box" and attempt to measure only the generated error, that error clearly the responsibility of the subroutine.

Error testing in general

The simplest and crudest type of error testing is a direct comparison of computed function values against published tables. Since this technique involves minimal effort, we might expect to obtain minimal assurances of quality as a result—and we do.

Such a comparison involves human handling of the standard, either by transcribing the table for keypunching or by visual comparison of computed results against tabular results. This imposes practical limitations on the number of tabular values used and on the credibility of the comparisons. It is far better to automate the process completely, i.e., to use the machine to compare computed function values against a machine-generated standard.

The sparseness of entries is a second objection to use of published tables. A subroutine may agree well with tabulated values and yet give very poor results for other arguments. Clearly, the best tests involve large numbers of arguments that are dense in some sense, and are not restricted to relatively small finite sets of "nice" arguments.

Finally, comparison against tables almost always involves conversion of tabulated decimal arguments into binary arguments. The consequent introduction into the arguments of the generated error from the conversion routine, the resulting transmitted error, and the subsequent error in conversion of computed results back to decimal form all contaminate the final error statistics. Unless the generated error of the routine under test is large, this method of testing will not be sensitive enough to detect it.

The Monte-Carlo approach using identities on the function[1] avoids two of the problems mentioned above. It involves large sets of random arguments and allows the machine to construct comparison values. However it is still difficult to design such a test which measures only inherent error. As an example, consider the testing of a cube root routine over the interval [1/8,1]. We might check the identity

$$x = (\sqrt[3]{x})^3 \qquad \tfrac{1}{8} \le x \le 1 \qquad (4.1)$$

or the identity

$$x = \sqrt[3]{x^3} \qquad \tfrac{1}{2} \le x \le 1 \qquad (4.2)$$

for several thousand random arguments. Consider (4.1) for the moment. Since the machine numbers form a discrete set, and the cube root is a contraction mapping of [1/8,1] onto [1/2,1], several different values of x must map onto the same value of the cube root. Cubing this result can return at most one of the original possible values of x. Testing based on (4.1) can thus be expected to indicate errors even when the subroutine computes the best possible machine number representation of the cube root. The identity (4.2) is much better, for it applies the cube root to the hopefully perfect cube of the original argument. Unfortunately, many machines introduce an error by failing to produce the most accurate representation of x^3.[5]

In this particular case, the effect of this argument error is probably minor, but even roundoff error in the computed argument can be disastrous. Consider, for example, the identity

$$e^{3x} = (e^x)^3$$

proposed in Reference 1 for checking the exponential routine. From (2.2) we find

$$\delta e^z = z\delta z.$$

If $3x \approx 10$, a rounding error of a half-unit in the computation of 3x is transmitted into 5 units error in e^{3x} independent of the accuracy of the exponential subroutine.

Additional care must be taken to choose identities which are independent of the analytic relations employed in computing the function. Tests based on the relation

$$\Gamma(1 + x) = x\Gamma(x),$$

which is almost universally used in the computation of the gamma function, may verify only that the relation has been correctly incorporated into the subroutine.

Bit pattern comparison

The scheme of error checking that we describe involves automatic tabular comparison where the standard table is generated within the machine. It requires the provision of a subroutine to compute standard values to a precision greater than that of the routine under test. Suppose for the moment that we are testing a single-precision routine. We must then provide a routine to compute the same function more accurately than single-precision. Full double-precision is not required, but the comparison routine must be more accurate than single-precision. Since final-bit accuracy is not required, the previously mentioned method of error checking using identities[1] can be used to verify the required accuracy of the comparison routine.

With such a comparison routine, it is an easy matter to construct a standard table of comparison values. We prefer to obtain dense sets of arguments for our tests by generating several thousand pseudo-random arguments over appropriate intervals. These arguments are computed in the test precision and extended to higher precision by appending appropriate low-order zeros. Computations can then be carried out with *identical* arguments in both the single and double-precision routines, eliminating the possibility of transmitted error, and a comparison of the results can be made. The details of this comparison will vary from machine to machine and individual to individual. We prefer to *round* (not truncate) the double-precision results to single precision and make a bit-pattern comparison of the results using fixed-point subtraction. We then tabulate the frequency of N-bit errors in the results for appropriate N. These statistics show how well the subroutine produces the machine number closest to the correct function value.

Additional statistics that are easily gathered and are of great interest are the maximum relative error

$$M = \max_{x_i} \left| \frac{F(x_i) - f(x_i)}{f(x_i)} \right| \tag{5.1}$$

and the root-mean-square relative error

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{F(x_i) - f(x_i)}{f(x_i)}\right)^2} \tag{5.2}$$

where F(x) is the test value and f(x) is the comparison value. The computation of these statistics is best carried out in the higher precision floating point arithmetic using the unrounded value of f(x). The results of extensive tests of this type are found in References 3,6.

Some of the details of the above procedure are subject to modification without destroying the validity of the process. The tests summarized in Reference 6, for example, included the unreported equivalent of bit pattern comparisons in the higher precision rather than the test precision. In terms of our example above, the single-precision result was extended to double-precision and the errors determined by a double-precision subtraction. The final tabulation of errors can then be interpreted in units of fractional parts of the least significant bit of the single-precision result. This approach allows some meaningful analysis of the inherited error for the second routine in the case of the composition of two function subroutines. For example, the computation of the logarithm of the gamma function might be accomplished by successively calling the gamma function and the logarithm subroutines. An intimate knowledge of the generated error from the gamma subroutine, which is the inherited error for the logarithm subroutine, allows an estimate of the final transmitted error.

Tests conducted at NASA Lewis Research Center[7] employed dense samples of evenly spaced, rather than random, arguments in addition to using the above modification. The appropriate tabulation of test results in this case provides valuable clues to troublesome argument ranges.

Other modifications are possible, and perhaps in widespread use. There are clearly cases where the relative errors used in (5.1) and (5.2) should be replaced by absolute errors by setting the denominators to unity.

Two critical matters are choice of test intervals and distribution of random arguments. These are intimately related to the internal structure of the subroutine under test. We feel there should be at least one separate test for each major computational path through the subroutine supplemented by special tests for a reasonably complete set of "critical ranges" of arguments, such as neighborhoods of points where the method of computation changes, regions where intermediate underflow or overflow may occur, etc. The importance of critical range tests cannot be overemphasized, for arguments causing a major breakdown in a routine are most likely to be found in fringe areas of computational abnormality. Clearly, intervals for which accuracy figures are not uniform should be subdivided. A uniform distribution of random arguments is adequate for most intervals, although an exponential distribution is often useful over extended ranges.[6]

The major problem in the use of this technique for checking double-precision routines is the provision of greater than double-precision comparison routines. We have faced this problem by either programmed triple precision or construction of comparison values on another computer with longer word-length.[3] This latter technique requires care to insure no contamination of arguments or function values in the transfer from one machine to another. Both techniques are slow, but tests need only be made once.

Generalizations

The error testing procedures described are easily generalized for testing of functions of two variables and functions of a complex variable. One major change is the selection of test arguments from a portion of a plane rather than from an interval. Regularly shaped regions, such as rectangles or circles, can usually be used. Many different combinations of uniform and exponential distributions of random arguments are possible for such regions. For example, tests using complex arguments uniformly distributed in amplitude and exponentially distributed in magnitude are reported in Reference 6.

The methods of error determination described above can be used whenever the function value is a real number. For complex-valued functions, the real and imaginary components of the function value can be tested separately as real functions of a complex variable. Bit pattern comparisons, maximum relative errors and root-mean-square relative errors can then be determined for each component as in Reference 3. But only the maximum relative error and root-mean-square relative error can be determined when the function value is considered as one complex number. The viewpoint taken in testing should reflect the method of computation within the subroutine, i.e., should depend upon whether or not the real and imaginary components of the function value are computed independently. This distinction was not made in Reference 3.

Table I presents the results of tests like those described, applied to Argonne National Laboratory's subroutine ANL B457S for computing x^y in single-precision floating point on the IBM System/360. Note in particular that the magnitude of the inherent error is such that it would be greatly distorted by even the smallest transmitted error introduced in the testing process.

Table I

TEST RESULTS FOR ANL B457S X**Y SUBROUTINE

Argument Range		Accuracy for Random Arguments Frequency of N-bit Error				Maximum Relative Error	RMS Error
X	Y	0	1	2	3		
(1/16,16)	(-4,4)	1933	67	0	0	$.456 \times 10^{-6}$	$.119 \times 10^{-6}$
$(2^{-16},2^{16})$	(-16,16)	1771	226	3	0	$.483 \times 10^{-6}$	$.124 \times 10^{-6}$
$(2^{-32},2^{32})$	(-8,8)	1906	94	0	0	$.459 \times 10^{-6}$	$.116 \times 10^{-6}$
$(2^{-64},2^{64})$	(-4,4)	1938	62	0	0	$.442 \times 10^{-6}$	$.111 \times 10^{-6}$
$(2^{-8},2^{8})$	(-32,32)	1609	347	44	0	$.561 \times 10^{-6}$	$.136 \times 10^{-6}$
$(2^{-4},2^{4})$	(-64,64)	1392	440	140	28	$.633 \times 10^{-6}$	$.175 \times 10^{-6}$

Additional tests

Accuracy tests should be supplemented by other simple tests to round out quality testing. The first of these is a timing check. It is a simple matter, if an internal clock is available, to time a subroutine for several thousand random arguments using a loop of some sort. The overhead for the loop can be obtained by timing an identical loop with the test subroutine replaced by a special subroutine whose only executable instruction is a return to the calling program. Such timing tests should be made for every major computational path through the subroutine. Frequently it is possible to replace the random argument timing test by one involving a fixed argument cycled through the routine several thousand times. It is important to loop enough times to minimize the effect of the coarseness of the clock. On an IBM 360/50, for example, the standard clock is incremented in units of 1/50 or 1/60 second while subroutines typically execute in less than a millisecond.[3]

Other obvious tests are checks of the error returns by using arguments at and just beyond the limits of acceptability. For subroutines claiming to accept all machine-expressible arguments this test should include extremely small and extremely large numbers to check for underflow and overflow problems.

SUMMARY

We have argued that quality testing of a function subroutine involves a great deal more than a quick accuracy check against published tables. We have described a technique for accuracy testing which is simple, is carried out completely in the machine, involves large numbers of arguments, and measures only the inherent error of the subroutine under test. Additional tests necessary for certification of function subroutines have been suggested. Finally, good quality tests are worth good documentation.

REFERENCES

1 C HAMMER
Statistical validation of mathematical computer routines
Proc S J C C 1967

2 H KUKI
Performance criteria for function programs of a single variable
Evaluation Guidelines, SHARE Numerical Analysis Project
SHARE Secretary Distribution 150 May 1966 Item C–4304

3 N CLARK W J CODY K E HILLSTROM
E A THIELEKER
Performance statistics of the Fortran IV(H) library for the IBM system/360
Report ANL 7321 Argonne National Laboratory 1967

4 U-Correspondence SHARE Secretary Distribution

5 W J CODY
The influence of machine design on numerical algorithms
Proc S J C C 1967

6 *IBM system/360 operating system Fortran IV library subprograms*
IBM System Reference Library C28–6596–1

7 L R TURNER
Private Communication

Towards an abstract mathematical theory of floating-point arithmetic *

by DAVID W. MATULA

Washington University
St. Louis, Missouri

INTRODUCTION AND SUMMARY

The representation of integers by a place value sequence of digits taken from a radix polynomial is one of the profound mathematical developments of all times. It provides an elegant foundation for mathematical questions where algorithmic procedures are needed, such as how to computationally effect the operations of addition, subtraction, multiplication, division and comparison. Nevertheless, as invaluable as the place value representation system is to computational mathematics, most questions regarding the mathematical structure of the integers are usually answered with proofs which make no reference to any representational form of the integers, but are based solely on an abstract characterization of the integers.

For example the existence of an infinite number of primes (a fact known in Euclid's time) is proved[1] by contradiction simply by considering what could be the factors of the number which would be the successor of the product of all primes if there were only a finite number of primes. Also the fundamental theorem of arithmetic, which states that all integers have a unique prime factorization, is readily established[1] by mathematical induction, as are many important structural questions about integers. The tool of mathematical induction is effective since its methodology, which is to prove a statement about integers by proving that it must hold on the successor integer to the set on which it is assumed to have been established, parallels the abstract recursive characterization of the integers whether by Peano's postulates or by the set theoretic hierarchy of ordinal numbers.

If we extend this discussion beyond integers to the rational numbers, again we see that the representational notation of a repeating decimal can be valuable for effecting efficient computational algorithms. But if we consider the question "Is the square root of 2 rational?" a procedure which attempts to find the actual representation of $\sqrt{2}$ and seeks to prove that it is a repeating decimal is cumbersome and illconceived. Instead, if we utilize the abstract characterization of rationals as ratios of integers, then the aforementioned question is equilvalent to asking for an integral solution of $a^2 = 2b^2$, and from known number theoretic results on the structure of the integers this is immediately seen to be impossible.

The point we wish to emphasize by the discussion of the preceding paragraphs is that the raison d'etre of the representational notation for integers and rational numbers is to effect efficient algorithms for computation, whereas the abstract characterization is invaluable for theoretical purposes.

Within most computers the scientific calculations are usually effected not with integers or rational numbers, but rather in a system of floating point numbers, which will herein be referred to as an *F. P. system.* These numeric systems are relatively new to mathematics and practically all information with regards to both the theory and computational application of F.P. systems is couched in a representational notation. This is not unexpected since the development and utilization of floating point numbers evolved to fulfill certain necessities of computerized computational requirements rather than by any intrinsic mathematical interest in the structure of such numeric systems. However, with over a decade of computational experience behind us, the need for a thorough understanding of the mathematical structure of these numeric systems

*This research was partially supported by the Advanced Research Projects Agency of the Department of Defense under Contract SD-302.

is ever more apparent. This need is not based on any esthetic concern purely to study and understand the structure of F.P. systems, but rather on the hard reality that many underlying computational difficulties can be attributed directly to the *structure* of F.P. systems. With no extant mathematical theory of F.P. systems available, and with most discussions of floating point numbers couched in a representational notation ill suited for theoretical investigation, many important questions which are raised in the introductions of texts on numerical analysis are often grouped under the term "roundoff error" and receive little further definitive consideration.

It is perhaps surprising that computational mathematicians, working within the "exact science" of mathematics, have remained oblivious to a serious number of theoretic and algebraically based study of the mathematical structure of F.P. systems. Thus for example while it is well-known that the failure of multiplication and division to be closed operations in F.P. systems leads to different context dependent meanings for the multiplicative operator, most discussions merely point out that these "pseudo multiply" and "pseudo divide" operations mean that F.P. systems are *not* algebraic fields. No positive attempt is made to study what these systems *are*. Furthermore, many computational problems involve finding a root of an equation $f(x) = 0$, thus it seems that more attention should be devoted to the study of F.P. systems so as to ascertain when such roots exist! The fact that the computed value of $f(x)$ may depend on the order of execution of the arithmetic operations composing $f(x)$ makes this a difficult but therefore still more essential area for serious mathematical investigation.

It is well-known that an abstract characterization of floating point numbers places the study of this class of numbers within the field of number theory. Many important questions relative to F.P. systems can thus be attacked by applying the wealth of results available in number theory. In the next section we develop such a characterization of floating point numbers, for which we prefer the term digital numbers, and spaces of these digital numbers which are known as significance spaces.[2,3] Some suggestive examples are given which demonstrate the power of the application of number theoretic results to yield theorems relevant to the structure of F.P. systems.

Since any computational number system in a computer is limited to a finite set of representable numbers, floating point numbers have been favored since on the one hand, they allow convenient algorithms for effecting numeric operations, and, on the other hand, they are reasonably conveniently spaced, so that the real numbers expected to be encountered can all be adequately approximated and resultant calculations generally closely represent calculations with real numbers. A full understanding of the structure of F.P. systems then must, for our purposes, provide a precisely defined relationship with the real number system. In the third section the relationship of floating point numbers to the reals is established through the definition of conversion mappings. It is then pointed out how a broad view of these mappings allows one to formulate a valid mathematical expression corresponding to any floating point computational expression without recourse to pseudo airthmetic operations or "ϵ" error terms.

Finally, in the last section, selected topics concerning numerical difficulties with F.P. systems are presented in this new mathematical formulation.

Digital numbers and significance spaces

A floating point number is typically represented by a finite sequence of digits to an assumed base along with an integral exponent for locating the "decimal" or radix point. If we assume the radix point is at the right of the sequence of digits, then a floating point number is of the form $\alpha\beta^\tau$, where α and τ are any positive or negative integers and β is a fixed base. The dependence of the floating point number on the choice of base is very important, since a real number representable by a finite number of digits in one base does not necessarily have a terminating expansion in another base. For the purposes of abstract characterization we prefer the term digital number to the term floating point number.

Definition: The real number x is a *digital number to the base* $\beta \geq 2$ if and only if $x = \alpha\beta^\tau$ for some integers α, τ. The *β-digital numbers*, D_β, is the set of all digital numbers to base β. It is evident that $x \epsilon D_\beta$ for some β if and only if x is a rational number, so that the β-digital numbers for $\beta = 2,3, \ldots$ provide subclassifications of the rationals. It should be noted however that the sets D_β and D_γ are not disjoint for $\beta \neq \gamma$, since every pairwise intersection contains at least all the integers. With these preliminaries a framework has been provided for discussing the possibilities of exact conversion of floating point numbers between various bases.

Exact floating point base conversion

In working with binary and decimal numbers it is readily observed that any binary integer or fraction can be represented as a terminating decimal. However the reverse does not hold, since 0.2 in decimal may be easily verified to have an infinite binary expansion. Thus in our abstract notation these observations mean that $D_2 \subset D_{10}$, $D_{10} \not\subset D_2$. It is therefore interesting to speculate on whether an F.P. system with a given base β will have each floating point number exactly representable by a finite number of digits to some other base γ.

Theorem 1: $D_\beta \subset D_\gamma$ if and only if $p|\beta$ (p divides β) implies $p|\gamma$ for all primes p.

Proof: Assume $p|\beta$ implies $p|\gamma$ for all primes p.

Let $x \epsilon D_\beta$, then $x = \alpha\beta^\tau$ for some integers α, τ. We may assume $\tau < 0$, for otherwise x is an integer and is therefore in D_γ. If m is the highest power of any prime which divides β, then from the assumption, $\beta|\gamma^m$. Thus $\beta = \gamma^m/k$ for some integer k, and $\beta^\tau = k^{-\tau}\gamma^{\tau m}$. Therefore $x = \alpha k^{-\tau}\gamma^{\tau m}$, and since $\tau < 0$, $\alpha k^{-\tau}$ is an integer so that $x \epsilon D_\gamma$.

To complete the theorem assume there exists a prime q such that $q|\beta$, $q \nmid \gamma$. Then clearly $q^{-1} \epsilon D_\beta$, and if $q^{-1} \epsilon D_\gamma$ then $q^{-1} = \alpha\gamma^{-\tau}$ must hold for some positive integers α and τ. Thus $\gamma^\tau = \alpha q$ so that $q|\gamma^\tau$, and since q is prime, $q|\gamma$, a contradiction. Therefore q^{-1} D_γ and $D_\beta \not\subset D_\gamma$.

Since any computational F.P. system involves a finite set of digital numbers to some base β, it is clear that another F.P. system with base γ can absorb exactly by conversion the original F.P. system if and only if both (1) γ contains all of the distinct prime factors of β, and (2) the new F.P. system has a sufficient number of digits.

To consider the question of whether two F.P. systems can be truly equivalent we first note the following general result which is immediately obtained from Theorem 1.

Corollary: $D_\beta = D_\gamma$ if and only if the condition $p|\beta \Longleftrightarrow p|\gamma$ holds for all primes p.

Although it is evident from the binary representational notation that every octal number with a finite number of digits can be represented as a terminating hexidecimal number and vice versa, the non trivial observation that the base 12 and base 18 digital numbers are the same follows immediately from this corollary.

However, it should be stressed that two different numbers with the same number of digits base 12 may require different numbers of digits base 18, and vice versa, so we cannot yet make any conclusions about whether two differently based F.P. systems can contain exactly the same set of real numbers.

Hence even though knowledge of the structure of floating point number systems can be gained by looking at the sets D_β they do not constitute suitable characterizations of floating point number systems since there is no bound on the number of digits in each digital number of D_β nor on the magnitude of these numbers. In fact, D_β is dense in the real numbers for any β, that is, there is an infinite number of members of D_β in any arbitrarily small interval of the real line. The next step towards characterizing a system of floating point numbers is to limit the number of digits in each digital number allowed in the system. This has been accomplished by the definition of a significance space.[2,3]

Definition: For the integers $\beta \geq 2$, called the *base*, and $n \geq 1$, called the *significance* (or precision), let the *significance space*, S^n_β, be the following set of real numbers: $x \epsilon S^n_\beta$, if and only if $x = \alpha\beta^\tau$ for some integers α, τ where $|\alpha| < \beta^n$.

Thus the significance space, S^n_β, is seen to be composed of certain digital numbers to the base β, and, in particular, the constraint $|\alpha| < \beta^n$ for the number $x = \alpha\beta^\tau$ assures us that the integer portion, α, can be represented by n digits to the base β and a sign. Note that since a significance space does not imply a bound on the exponent part, τ, of $x = \alpha\beta^\tau$, S^n_β still contains an infinite number of elements. However, any closed interval of real numbers disjoint from zero contains only a finite number of elements of S^n_β. A further constraint bounding the exponent part of each digital number would fully characterize a system of floating point numbers, however, the significance space as just defined is a more tractable mathematical structure to analyze. Furthermore the significance space is the appropriate tool for understanding the more perplexing computational numeric difficulties, specifically those which are *not* simply attributable to underflow and overflow.

Although most of the theoretical results which are relevant to computational difficulties in F.P. systems must await the definition of conversion mappings of the next section, an interesting example relating a theoretical property of significance spaces to a computational problem with floating point numbers is contained in the following.

Tape updating problem[4]

Suppose a floating point number is read in from a B.C.D. coded tape and converted to binary for internal machine reference and then reconverted to decimal for preparing the new updated output tape. One would like to believe that any such "constant datum" which is never changed in the successive tape updatings will remain invariant despite the frequency of updating. However, a little reflection suggests that the successive conversions of this datum may cause the value of this constant to drift, especially if all conversions are effected by truncation rather than rounding. The following observation relates this practical problem to a theoretical question about significance spaces.

Successive truncation conversions between two differently based F.P. systems monotonically decreases the magnitude of a number until a real number exactly representable in both F.P. systems is encountered. This suggests looking at the intersection of the corresponding two significance spaces, with the result[4] that for the bases 2 and 10 with fixed numbers of digits n and m respectively, the set $S_2^n \cap S_{10}^m$ has only a finite number of elements.

In particular the minimum strictly positive value in $S_2^{24} \cap S_{10}^7$ is 2^{-10}. Thus returning to the tape updating problem, if a quantity $y < 2^{-10}$ appears on a 7-significant digit B.C.D. tape which is updated on a binary machine with truncation conversions in both directions, then the quantity y will monotonically decrease, thereby losing all accuracy, and eventually underflow.

In conclusion, the abstract mathematical characterization of floating point numbers is seen to be a tool having great potential for furthering our understanding of the actual difficulties encountered in practical computerized numeric calculations.

The generalized conversion mapping

Since only a finite set of real numbers can be represented exactly in an F.P. system, the element of "round-off error" enters when we encounter a numeric quantity which is not one of the finite set of representable numbers. This phenomena can be encountered in two ways; (1) If input data are not in the appropriate base a conversion is necessary and the converted number may not equal the original number, (2) all four arithmetic operations, addition, subtraction, multiplication and division, are not closed operations within a system of floating point numbers so an arithmetic operation may only approximate the correct result.

These two kinds of error are usually treated separately, the first being largely ignored since I/O is generally non-recurrent allowing only one small error to enter from this source. The latter error is generally described in representational notation either by (1) describing the full set of digits that would occur in the exact operation or by (2) storing an appropriate remainder term in a register, and then specifying whether the computer hardware utilizes the additional information to round or truncate to an approximate result.

Note that a common aspect to both of these phenomena is that a real number is encountered that is not exactly representable by the fixed n digits to the base β available for a specified set of machine representable numbers. Both approximations may be made explicit analytically by introducing the notion of a generalized conversion mapping whose domain is the real numbers and whose range is the significance space S_β^n. As we are interested in both rounding and truncation procedures, the following two generalized conversion mappings[2] provide analytic characterizations of these approximation procedures.

Definition: The truncation conversion mapping, T_β^n, and the rounding conversion mapping, R_β^n, of the real numbers into S_β^n are defined, for $n \geq 1$, $\beta \geq 2$, by:

Truncation conversion:

$$T_\beta^n(x) = \begin{cases} \max\,\{y \mid y \leqq x,\ y \in S_\beta^n\} & \text{for } x > 0, \\ \min\,\{y \mid y \geqq x,\ y \in S_\beta^n\} & \text{for } x < 0, \\ 0 & \text{for } x = 0. \end{cases}$$

Rounding conversion:

$$R_\beta^n(x) = \begin{cases} \min\,\{y \mid x < (y+y')/2,\ y \in S_\beta^n\} & \text{for } x > 0, \\ \min\,\{y \mid x \leqq (y+y')/2,\ y \in S_\beta^n\} & \text{for } x < 0, \\ 0 & \text{for } x = 0, \end{cases}$$

where y' is the *successor* of y in S_β^n defined for $y \neq 0$ by $y' = \min\,\{z \mid z > x,\ z \in S_\beta^n\}$.

Note that a distinction is made in the definition of the mappings of positive and negative numbers so that $T_\beta^n(-x) = -T_\beta^n(x)$ and $R_\beta^n(-x) = -R_\beta^n(x)$.

Utilizing these definitions for describing input conversions by truncation to an F.P. system with six significant digits base 16, one may simply state that the decimal numbers 2.594, 3.5×10^6 and the binary number 11.1_2 become T_{16}^6 (2.594), T_{16}^6 (3.5×10^6) and T_{16}^6 (11.1_2) respectively. This notation is self contained and unencumbered by any additional description of the actual method of representing decimal and binary numbers in a hexidecimal system. Turning to the precise characterization of arithmetic operations in F.P. systems, rounded multiplication of x and y in the above F.P. system is succinctly denoted by R_{16}^6 $(x \cdot y)$, where $x \cdot y$ has the usual meaning and no "pseudo multiplication" symbol need be introduced. This procedure is readily extended to precisely describe whole arithmetic expressions. For example, if X, Y, and Z are input values which are truncated into an F.P. system of six digits base 16, and if the following FORTRAN expression using X, Y, and Z is executed in this F.P. system utilizing rounded arithmetic,

$$X * Z * * 2 - \mathrm{SQRT}(Y + 1000. * X)/2. \tag{1}$$

then a mathematical characterization of this expression is given by the following formulation:

$$R^6_{16}(R^6_{16}(T^6_{16}(X)\cdot R^6_{16}([T^6_{16}(Z)]^2)) - R^6_{16}(R^6_{16}(\sqrt{R^6_{16}(T^6_{16}(Y) + R^6_{16}(1000T^6_{16}(X)))}/2.)) \tag{2}$$

The latter expression (2) is illustrative of how a computerized calculation can be abstractly defined, however, we do not suggest that such unwieldy expressions need often be written. Rather, if f(x) is the desired function of x which we wish to calculate on a computer, then we can let f*(x) refer to the appropriate function of the form of (2) for computational analysis purposes.

The collection of such functions available for computerized calculations can be characterized as follows.

Definition: For $n \geq 1, \beta \geq 2$, the floating point rounded n-digit base β digital functions, denoted F^n_β, are exactly those functions generated by the following rules:

1. Constant Functions:

$$c \in S^n_\beta \Rightarrow c \in F^n_\beta$$

2. Primitive Variables:

$$x \text{ a variable with domain } S^n_\beta \Rightarrow x \in F^n_\beta$$

3. Composed Functions:

Let $f(x_1, x_2, \ldots, x_k), g(y_1, y_2, \ldots, y_m) \in F^n_\beta$ be functions of k and m variables respectively, then

$$h(x_1, \ldots, x_k, y_1, \ldots, y_m) = R^n_\beta(f(x_1, x_2, \ldots, x_k) + g(y_1, y_2, \ldots, y_m)) \in F^n_\beta$$

$$h(x_1, \ldots, x_k, y_1, \ldots, y_m) = R^n_\beta(f(x_1, x_2, \ldots, x_k) - g(y_1, y_2, \ldots, y_m)) \in F^n_\beta$$

$$h(x_1, \ldots, x_k, y_1, \ldots, y_m) = R^n_\beta(f(x_1, x_2, \ldots, x_k) \times g(y_1, y_2, \ldots, y_m)) \in F^n_\beta$$

$$h(x_1, \ldots, x_k, y_1, \ldots, y_m) = R^n_\beta(f(x_1, x_2, \ldots, x_k)/g(y_1, y_2, \ldots, y_m)) \in F^n_\beta$$

where appropriate modification of the arguments of the function h is assumed if any of the variables x_i are identical to any of the variables y_j.

Thus F^n_β includes mathematical characterizations of the computerized arithmetic expressions which are available on an n-digit base β machine with rounded arithmetic excluding underflow and overflow considerations (the case of truncated arithmetic may be treated similarly). A computerized version of f(x) then is an approximation in function space, with some function $f^*(x) \in F^n_\beta$ approximating f(x). Note that F^n_β effectively includes the special functions (exponential and trigonometric) since these are generally calculated on machines via polynominal or repeated fraction approximations, however the function resulting from pointwise numerical integration of a function $f^*(x) \in F^n_\beta$ would generally not be in F^n_β, since inequality side constraints are used and this is not part of the procedure for creating functions of F^n_β. The utility of this definition of F^n_β is that the theoretical questions about the properties of the class of expressions which can be executed in the floating point hardware of the computer can be abstracted to the level of investigating properties of F^n_β.

In summary the generalized conversion mapping stands out as the important analytic tool which allows us to abstractly characterize both base conversion mappings and the functions equivalent to computerized floating point realizations of arithmetic expressions.

The analysis of floating point number systems

It will be instructive to demonstrate the formulation of questions relevant to the structure of floating point number systems in terms of the analytic structures we have just introduced. The following five topics are not intended to be exhaustive but do cover a wide range of numerical questions relevant to floating point arithmetic.

It is not our intention to suggest that the new formulations of these problems will provide their immediate resolution, rather we propose that these formulations provide the precise mathematical characterizations which are the necessary first step in any serious attempt to develop a general theory of floating point arithmetic.

Errors of base conversion

In an earlier section the tape updating problem pointed out one difficulty with decimal to binary base conversion under truncation. Another question relevant to base conversion is: "how many digits should be available in the computer's floating point numeric base to maintain the distinctness of different input quantities of a fixed number of decimal digits?" This question may be formulated mathematically as: "Given $m \geq 1$, and $\beta \geq 2$, what is the minimum value of n such that the restricted mapping $R^n_\beta : S^m_{10} \rightarrow S^n_\beta$ is a one-to-one mapping of S^m_{10} into (but not necessarily onto) S^n_β?" In the solution[2] to this problem it should be noted that although elementary inequality conditions are adequate to establish a number of digits, n, which is sufficient to insure a one-to-one mapping, the necessity condition required to verify the minimum value of n was found by applying Kronecker's Theorem[5] from number theory to the appropriate number theoretic formulation of this problem.

Proceeding further, the question of whether an iterated in-and-out conversion can always identically return the original number is formulated by asking for those conditions on n, β, m, ν such that $x = R^n_\beta(R^m_\nu(x))$ for all $x \epsilon S^n_\beta$. This problem was completely resolved[3] by the same methods referred to above. It is instructive to compare the general results of this abstract treatment[3] with a previous paper[6] which utilized representational notation to establish only the simpler part of the general theorem on in-and-out conversion.

Algebraic structure of floating point calculations

Arithmetic calculations with floating point numbers are intended to provide a realizable computational process resembling operations in the field of real numbers. However, quite apart from the approximations necessitated when calculating with floating point numbers, the invariance of the value of arithmetic expressions under the familiar procedures of rearrangement and cancellation is not obtained. Thus the algebraic structure of floating point calculations is also only an approximation to the algebraic structure of the reals. This fact has been pointed out many times by showing that the pseudo-operations of addition and multiplication realized in floating point calculations do not possess the structure of a field.

Without appealing to pseudo operations, where the addition and multiplication symbols must have meanings context dependent on the representational form of the numbers on which they operate, the questions relevant to field theoretic structure can be succintly stated utilizing significance spaces and conversion mappings so as to preserve the normal meaning of the arithmetic operators in the real number system. In the following we formulate all the questions in our notation and state the answers without proof, as they are generally well-known and in any case represent instructive exercises.

Does Floating Point Arithmetic Satisfy the Axioms of a Field?

If for $x, y \epsilon S^n_\beta$, we declare floating point addition, subtraction, multiplication and division of x and y to be given by $R^n_\beta(x+y)$, $R^n_\beta(x-y)$, $R^n_\beta(x \cdot y)$ and $R^n_\beta(x/y)$ respectively, then we have for:

F1. Closure under addition:
Is $R^n_\beta(x + y) \epsilon S^n_\beta$ for all $x, y \epsilon S^n_\beta$?
Answer: Yes

F2. Closure under multiplication:
Is $R^n_\beta(x \cdot y) \epsilon S^n_\beta$ for all $x,y \epsilon S^n_\beta$?
Answer: Yes

F3. Associativity of addition:
Is $R^n_\beta(x + R^n_\beta(y + z)) = R^n_\beta(R^n_\beta(x + y) + z)$ for all $x, y, z \epsilon S^n_\beta$?
Answer: No

F4. Associativity of multiplication:
Is $R^n_\beta(x \cdot R^n_\beta(y \cdot z)) = R^n_\beta(R^n_\beta(x \cdot y) \cdot z)$ for all x, y, $z \epsilon S^n_\beta$?
Answer: No

F5. Commutativity of addition:
Is $R^n_\beta(x + y) = R^n_\beta(y + x)$ for all $x, y \epsilon S^n_\beta$?
Answer: Yes

F6. Commutativity of multiplication:
Is $R^n_\beta(x \cdot y) = R^n_\beta(y \cdot x)$ for all $x, y \epsilon S^n_\beta$?
Answer: Yes

F7. Identity under addition:
Does there exist $x \epsilon S^n_\beta$ such that $R^n_\beta(x + y) = R^n_\beta(y)$ for all $y \epsilon S^n_\beta$?
Answer: Yes (zero)

F8. Identity under multiplication:
Does there exist $x \epsilon S^n_\beta$ such that $R^n_\beta(x \cdot y) = R^n_\beta(y)$ for all $y \epsilon S^n_\beta$?
Answer: Yes (unity)

F9. Inverses under addition:
For each $x \epsilon S^n_\beta$ does there exist $y \epsilon S^n_\beta$ such that $R^n_\beta(x + y) = 0$.
Answer: Yes

F10. Inverses under multiplication:
For each $x \epsilon S^n_\beta$ does there exist $y \epsilon S^n_\beta$ such that $R^n_\beta(x \cdot y) = 1$
Answer: No.

Note: Question F10 has many interesting sidelights and has been the subject of a rather extensive investigation.[7] Thus, for example, in S^3_{10} the number 3 has the inverse .334, yet $R^3_{10}(1/3) = .333$ is not an inverse.

F11. Distributivity:
Is $R^n_\beta(x \cdot R^n_\beta(y + z)) = R^n_\beta(R^n_\beta(x \cdot y) + R^n_\beta(x \cdot z))$ for all $x, y, z \epsilon S^n_\beta$?
Answer: No

The failure of many of the axioms of a field means that many different functions can be suitable candidates for approximation of the single real function f(x), and study should be given to determine if a best choice is possible.

Roots of equations

A general mathematical problem is to find a root of the equation $f(x) = 0$, and when f(x) is suitably behaved it is assumed that this is not difficult to solve via algorithmic procedures with real arithmetic. However numerical work in evaluating f(x) is actually performed with a specified approximation $f^*(x) \epsilon F^n_\beta$, and it appears pertinent to investigate the question of whether $f^*(x) = 0$ has any roots $x \epsilon S^n_\beta$.

Letting $f(x) = ax + b$ be a simple linear equation with $a, b \epsilon S^n_\beta$ and choosing $f^*(x) = R^n_\beta\ (R^n_\beta(a \cdot R^n_\beta(x)) + b) \epsilon F^n_\beta$, it is observed that a root of $f^*(x) = 0$ must satisfy $R^n_\beta\ (a \cdot R^n_\beta(x)) = -b$.

For the case $b = -1$ this corresponds to finding an inverse of a in S^n_β and it has already been pointed out that such inverses need not exist, and, even when they do, they are not necessarily given by $R^n_\beta\ (1/a)$. The actual solution of $R^n_\beta\ (a \cdot R^n_\beta(x)) = -b$ necessitates solving a Diophantine equation subject to an inequality constraint, and there is small utility in pursuing this course.

However two pertinent observations emerge from this discussion. (1) $f^*(x) = 0$ need not have a solution even though $f(x) = 0$ has, so that iterative schemes which attempt to find a root of f(x) by working with $f^*(x)$ must be designed with this knowledge in mind. (2) $f^*(x)$ is a piecewise continuous step function of the real variable x, thus $f(x) - f^*(x)$ is a piecewise continuous function in x giving the error in the computerized realization of f(x), and it is this well defined function which should be scrutinized for error bounds in applied numerical analysis.

Integration and differentiation of f* (x)

The erratic behavior of a digital function $f^*(x)$ as a function of x is a phenomena of immense importance to numerical integration and differentiation. For our purpose, a typical numerical integration procedure for determining $\int_1^2 f(x)\,dx$ will divide the region 1 to 2 into N parts, evaluate $f^*(x)$ at these N mesh points and average them, and then proceed through the same procedure at 2N points (halving the interval), and compare the results to see if a suitably small difference is attained. It is evident that as this halving procedure is continued one generally will converge towards $\int_1^2 f(x)\,dx$ as long as the $2^k N$ mesh points at which $f^*(x)$ is evaluated is much smaller than the $\beta^{n-1}+1$ points of S^n_β in the interval [1,2]. However as the number of halving procedures increases so that $2^k N$ approaches $\beta^{n-1}+1$, the integration procedure will begin to add significance to the erratic behavior of $f^*(x)$, and successive halvings may well determine a result deviating more from $\int_1^2 f(x)\,dx$ than previous results. Furthermore, increasing the halving procedure so that $2^k N < \beta^{n-1}+1$ does not generate any new points but merely duplicates points already calculated, and it should be evident that the consequent convergence of the numerical procedure does not in any way substantiate the calculated result as a meaningful estimate of $\int_1^2 f(x)\,dx$.

In performing numerical integration one must tread his way along the same path as the physicist performing the classical derivations of thermodynamics, where each of the infinitesimal intervals on which the limiting procedure of the calculus depends must still be large enough to allow a sufficient number of atoms so that the statistical behavior of the assemblage of atoms is meaningful. Thus the numerical analyst is obliged to verify convergence of the integration when $2^k N \ll \beta^{n-1}+1$, otherwise the results should be considered spurious.

To appreciate the difficulties possible in numerical differentiation observe that $f(x) = x/(x+2)$ is a monotonically strictly increasing function of x for positive x, yet $f^*(x) = R^n_2(x/R^n_2(x+2))$ exhibits a very peculiar behavior. Note that as $x \rightarrow 1$, $x \epsilon S^n_\beta$, each successive x increases the numerator, yet the denominator, $R^n_2(x+2)$, will not change with every successive increasing value of $x \epsilon S^n_\beta$ in this range, and when it does change the proportional increase in the denominator will be greater than in the numerator for x sufficiently close to unity. Therefore $f^*(x)$ shows an oscillating behavior as $x \rightarrow 1$, hence numerical differentiation schemes which utilize too fine a difference in x may pick up this spurious behavior of $f^*(x)$ and yield even the wrong sign for the derivative.

It should be pointed out that the stock answer of carrying more digits, that is, increasing n in the expression $f^*(x) = R^n_2(x/R^n_2(x+2))$, would certainly decrease the error function $f(x) - f^*(x)$, but does not alter the oscillatory behavior of $f^*(x)$ compared to the monotonic behavior of f(x).

Accumulated roundoff error

The accumulation of roundoff error is the most perplexing difficulty with computerized calculations, and no overall resolution of this problem is available. The simple propagative calculation of error bounds yield bounds which are generally too wide when the same variable is encountered more than once in a calculation procedure, since the correlation of the errors which are introduced is not reflected in the bounds. Thus if it is known that $a \leq f(x) \leq b$ and $2a \leq g(x) \leq 2b$, then clearly $2a - b \leq g(x) - f(x) \leq 2b - a$, whereas if actually $f(x) = x$, $g(x) = 2x$, then the sharp bounds would be $a \leq g(x) - f(x) \leq b$.

Statistical treatments of roundoff error have given useful heuristic bounds but the fundamental problem we wish to treat is deterministic. It is possible and convenient in our notation to treat certain controlled sequences of accumulating error. For example the treatment of accumulated conversion error has been fully described,[4] and it is suggested that the following kind of investigations of accumulated calculation error would be instructive.

Let $f(x) = x^k$, and define $f^*(x)$ as follows:

$$f_1(x) = R_\beta^n(x)$$

$$f_j(x) = R_\beta^n(x \cdot f_{j-1}(x)) \text{ for } 1 < j \leq k$$

$$f^*(x) = f_k(x)$$

A study of the behavior of $f^*(x)$ should yield some light on accumulated correlated multiplicative error. Similar studies with recursive division and with polynomial evaluation with all coefficients unity might provide valuable insight towards understanding the overall phenomena of accumulated roundoff error.

REFERENCES

1 W J LeVEQUE
Topics in number theory
Vol I Addison-Wesley Reading Massachusetts 1955

2 D W MATULA
Base conversion mappings
Proc S J C C 1967 Vol 30 311-318

3 D W MATULA
In-and-out conversions
C A C M Vol 11 No 1 January 1968 47-50

4 D W MATULA
Accumulated conversion error in mixed-base computation
J A C M (submitted for publication)

5 G H HARDY E M WRIGHT
An introduction to the theory of numbers
Oxford London England 1960 375

6 I B GOLDBERG
27 bits are not enough for 8-digit accuracy
C A C M Vol 10 No 2 February 1967 105-106

D W MATULA
The inverse(s) of a floating point number
In preparation

A panel session—Small computers for data terminal network control

GEORGE W. PATTERSON, *Chairman of Session*

Sanders Associates, Inc.
Nashua, New Hampshire

Small computers in data networks

by C. B. NEWPORT

Honeywell, Inc.
Framingham, Massachusetts

Small computers, costing between, say \$10,000 and \$50,000 each, are rapidly proving to be very important elements in data communications networks. The value of these machines lies in their high speed data manipulation capability rather than in their computing power; in fact, the direct arithmetic capability rarely exceeds simple binary addition and subtraction.

The uses of these small computers can be grouped into two main areas:

1. Remote message concentrators and terminal controllers, and;
2. Communication interfaces for larger machines.

The primary function of remote concentrators is to reduce line costs by multiplexing the data from many low speed lines, up to 150 Baud, on to one or more medium speed lines. This function, by itself, can be achieved by many hardwired devices as well as by a stored program computer; however, the use of a computer with significant storage and data manipulation capability immediately brings many other advantages. Data can be blocked before transmission over the medium speed line, thus, normally eliminating demultiplexing at the large computer site, code conversion and data editing can take place, and various terminal control functions such as automatic answering, polling and error control can be implemented quite flexibly.

The system that Honeywell has implemented in conjunction with American Airlines on the Sabre reservation system is a good example of both concentration and terminal control (see Figure 1).

DDP-516's with two Data Line Controllers interface to the two 2400 Baud lines used for data transmission and hub polling on the Sabre network. On the low speed side up to 60 IBM 1977 agents sets operating at 148.5 Baud are interfaced to the DDP-516 via a multi-line controller. The DDP-516 handles the functions of assembling data blocks, editing out meaningless blanks, responding to polling messages, and the generation and checking of error control information. A number of these remote concentrators have now been in operation for more than six months and they have demonstrated an improvement in response time of about 30 percent over the previous IBM 1006 hardwired terminal control

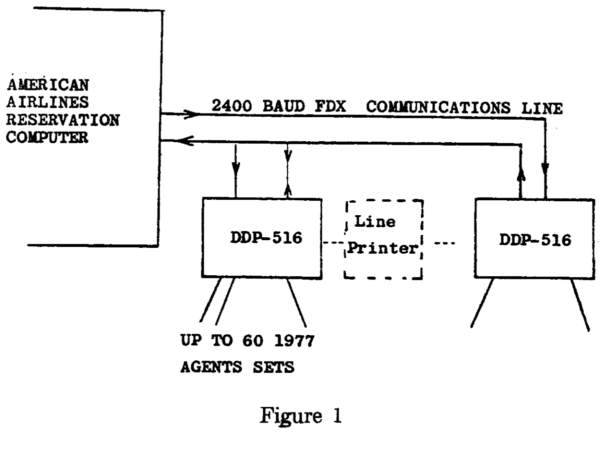

AMERICAN AIRLINES RESERVATION COMPUTER
2400 BAUD FDX COMMUNICATIONS LINE
DDP-516
Line Printer
DDP-516
UP TO 60 1977 AGENTS SETS

Figure 1

units. This is largely due to the data editing taking place in the DDP-516 so that redundant information does not have to be transmitted to Sabre or to the agents sets.

As an indication of the flexibility of the system it was found desirable to add a line printer since some of the reports required on a daily basis from Sabre took an inordinately long time to print on the IBM 1977, 15 character/second agents sets. The standard DDP-516 line printer was, therefore, added to the system, but since this is only an output device, it was not possible to treat it simply as a higher speed version of the 1977 terminal. The Sabre system always responds with a message directed to the terminal that made the request, so "output only" terminals are normally excluded. In this case, with the DDP-516 as the terminal controller, it is possible to monitor input messages from the appropriate agents sets and look for the code indicating that reply is required on the line printer. The message that is actually passed on to Sabrè is then a modified version of the request which makes it appear to Sabre as though it had come from the line printer. Sabre then makes its reply back to the apparent originating terminal as usual, and the print-out appears as required on the line printer. This kind of modification would be a major undertaking with a hardwired controller.

The price of the DDP-516 system, with 12K of core, was approximately half that of the corresponding hardwired system. Price comparisons are, however, misleading and need to be considered in each individual situation. In particular, the programming costs need to be considered in relation to the design costs of a hardwired system and general comparisons only tend to be useful when significant quantities are being considered, say, above 10 or 20, so that one-time costs become minimal.

The second use for small computers in data communications, interfacing to larger computers, is illustrated by a number of applications in which DDP-516's and 416's have been interfaced to IBM 360/50 and 360/67, Honeywell H-1200 and 2200 and, perhaps, more interestingly, large scale DDP-516 systems.

The Honeywell H-1648 Time-Sharing System (see Figure 2) illustrates the use of a 4K DDP-416 computer to provide the communications interface to a pair of 32K DDP-516's which are providing a time-sharing service for up to 48 simultaneous users. The two DDP-516's, the Control Computer and the Job Computer, are normally fully occupied providing the time-sharing and computing operations required by users, while the DDP-416 is dedicated to handling the communications lines. The control computer is the heart of the system and provides the terminal users with the ability to create and manipulate files on the disc

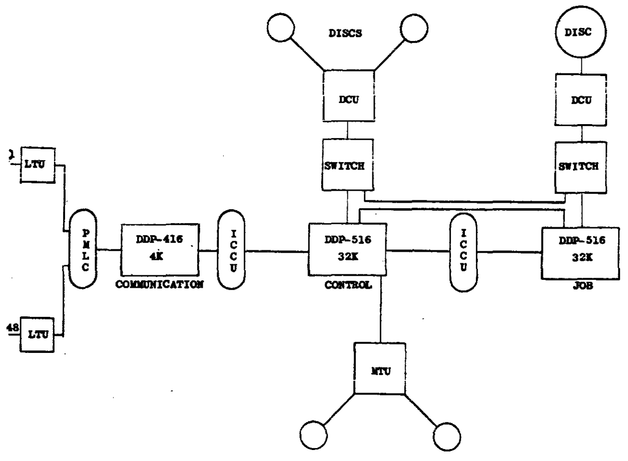


Figure 2

complex. When computing operations are required (rather than control or editing functions) such as compilation of a file or the running of an already compiled file, then the control computer schedules these tasks, provides core allocation and disc references for the job computer and initiates operation via the ICCU (Inter-Computer Communications Unit). The job computer allocates a predetermined time period (between 100 ms and one second) for the running of each job and if it is not completed within that time, swaps it out on to the disc and starts the next job. The I/O structure on the DDP-516 allows processing concurrently with transfers to and from the disc so that delays due to disc access time are not important.

The DDP-416 handles all communication functions and simply presents the control computer with strings of characters and an indication of the terminals from

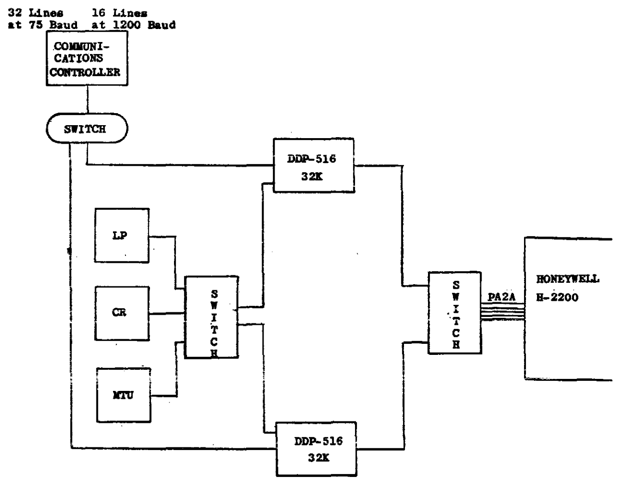


Figure 3

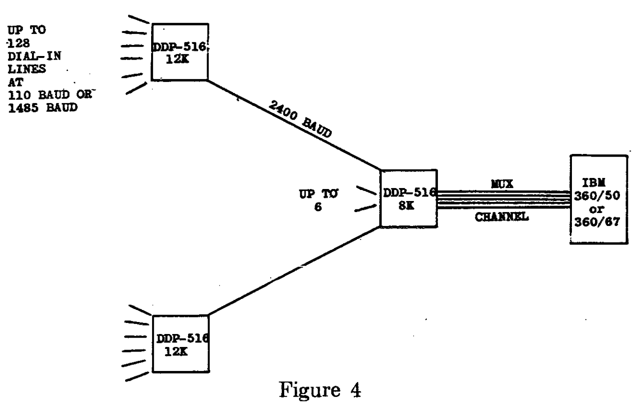


Figure 4

which they have been received. The DDP-416 is used as a bit sampler and is continually scanning the state of the incoming lines at eight times the bit frequency. The interface is designed to be able to receive signals from both dedicated lines and from the switched network. The samples of the incoming lines are processed to determine the state of the line, and if in the character mode, successive samples are used to assemble the character.

To achieve complete verification that the DDP-416 has received the character correctly echo-back checking is used. At each terminal the teletype keyboard and printing mechanisms are connected respectively to the transmit and receive sides of the full duplex communication line, and each character received by the DDP-416 is immediately echoed back to be printed. The user can then be sure that if the teletype is correctly printing the characters which he types, the characters will have been successfully received by the computers.

The DDP-416 is capable of handling at least 64 lines at 110 Baud and if a DDP-516 is used this can be expanded up to 128, although in this time-sharing application the remainder of the system is not designed to handle more than 48 simultaneous users.

The use of a small computer as the communications interface provides cost saving and an increase in flexibility over a hardwired interface. Perhaps, the most significant feature in a system that is subject to high peak loads is the ability of the communications processor to provide flexible buffer storage which can be dynamically allocated to the busy lines, and temporarily suspend the passing of data to the larger computer. In addition, if echo-back is used, the input from the terminal can be slowed down by simply delaying the echo-back. In a system with many random inputs, peak loads will only last for a very short time, so these buffering and delaying techniques are very useful in increasing ultimate system capacity without causing noticeable effect to individual users.

The use of a small computer as a terminal controller for a large computing system

by H. B. BURNER, R. MILLION, O. W. RICHARD and J. S. SOBOLEWSKI

Washington State University
Pullman, Washington

In the spring of 1967 Washington State University began investigating the possibility of replacing or supplementing its IBM 2702 terminal control unit with a small computer. We hoped to realize four significant advantages from such a move:

1. A small computer—being programmable—offered the opportunity of increased flexibility in the control of remote terminals.
2. The prices of small computers had decreased to the point that it seemed reasonable to expect such a system to cost less than the IBM 2702.
3. Since IBM required that each TTY line be connected to the 2702 via a 3233 line adapter and a 103A or 103F data set, we felt that significant savings might be obtained through lower incremental costs per line, achieved by providing either direct TTY connection or the use of lower cost data sets on leased private lines.
4. By handling as many communication problems as possible in the small computer, we hoped to reduce substantially the amount and complexity of system modifications in the large computer.

We should emphasize that our primary concern was the internal campus network which involves only local communication lines. There was no thought of using the small computer at a remote location as a data concentrator in order to minimize long distance line costs. This is, of course, an application of small computers to which many companies, particularly time sharing services, are addressing themselves.

During the past 18 months we have been working closely with the Interdata Company on a system designed to meet the above objectives. The system is being delivered in two stages, the first of which has been operating successfully for some months. The second stage is being installed at the time of writing.

Stage 1 consists of an Interdata Model 3 processor, a Multiplexor Control Unit and an interface to the IBM 2870 Multiplexor Channel. The Multiplexor Control Unit connects 32 low speed terminals to the I/O bus of the Model 3 through four data line units each of which by using appropriate couplers can accept data from or send data to 8 remote terminals via direct telegraph lines, TWX lines, and switched or private telephone lines.

To facilitate handling of the multiplexor data, Interdata has provided a special BIM (branch of multiplex) instruction implemented in the read-only memory of the Model 3. This instruction, which is executed as a result of a clock interrupt every 1/7 of a bit time, assembles and disassembles the characters bit by bit stripping or adding the start and stop bits. By interrupting 7 times per bit time, we are able to ensure a sample close to the center of each bit pulse. When a character is completely assembled, it is placed in a fixed location in core memory corresponding to the terminal from which it was sent. Similarly, on output a character is taken from a fixed location in memory and sent out to a given terminal.

The interface between the Interdata system and the IBM 2870 was designed initially to make the Interdata appear as much like an IBM 2702 as possible. Thus, for example, each terminal attached to the Interdata is treated as one of the possible 256 devices that can be attached to the 2870. Also, transmission to and from the 2870 is always in byte mode. However, as a result of early tests with the system, several changes were made to the interface including the addition of extra command and status bits to provide more control to the program operating in the Interdata. This enables us to use the system as a controller in either a polling or contention situation.

When operating in a polling environment, the program in the Interdata is controlled by the 360/67 via interrupts. When the 360 requests data from a given terminal, an input switch is set for that terminal. The program continually scans the one-character-per-terminal buffers for input. When input is available, that is, when a character has been typed, it is converted to 360 format and placed in a line buffer for the terminal provided the input switch is set. If a backspace or line delete character is received, the line is appropriately edited and when the terminating character (x- off) is typed, the edited line is sent to the 360.

When the 360 has data to be sent to a terminal, the data is read, converted, and placed in the line buffer for the terminal. In addition, an extra carriage return and line feed are inserted after the 72nd character to allow printing of full printer length lines. An output switch for the given terminal is then set. Once per character time, the output switch is checked. If set, a character is transferred from the line buffer to the one-character-per-terminal buffer for transmission to the terminal.

With this system all of our objectives have been realized. However, in an attempt to gain even greater flexibility and lower the per line cost, we plan to expand the system in January 1969 with the addition of a second processor—an Interdata Model 4. All of the processing with the exception of the BIM instruction will then be done on the Model 4, a processor which is seven to ten times as fast as the Model 3. The Multiplexor Control Unit and the read-only memory of the Model 3 will be altered to accommodate 64 lines, and we anticipate that virtually all of the available time on the Model 3 will be taken up in executing the expanded BIM instruction. However, there should be relatively little interference with the Model 4 so we expect to have available 15 to 20 times as much processing capability as we now have. Between 5 and 10 percent of this capacity will be used in servicing four 2400 band lines connected directly to the Model 4.

It is clear that a terminal controller such as we have described can simplify the organization and operation of the operating system within the primary computing system. Differences among terminal devices and codes can be accommodated within the controller so that all devices appear the same to the primary system. In addition, control characters such as backspace and shift can be recognized by the controller and appropriate editing action taken before the information is transmitted. In fact, we expect within the next few months to be able to provide immediate syntax checking of each statement of a FORTRAN program as it is entered from a terminal into our remote-job-entry batch processing system, thus providing one of the most useful features of conversational programming in an essentially batch processing environment.

A system for designing fast programming language translators

by VICTOR SCHNEIDER

University of Maryland
College Park, Maryland

INTRODUCTION

This paper demonstrates a straightforward algorithm for converting programming language grammars into pushdown-store automata translators. The language grammar is written as a "translation grammar" in which, for each syntactic rule, there is a corresponding "rule of translation" that recursively specifies the reverse Polish string translation of the objects in the syntactic rule. This augmented grammar is transformed directly into a flow chart for the appropriate translator.

To prevent the necessity of backtracking during translation, an algorithm is presented for converting nondeterministic translators into deterministic (i.e., single-scan) translators. Those languages for which this backtracking elimination algorithm fails contain as a subset the ambiguous programming languages.

From considerations of machine topology, upper bounds on memory storage requirements and computational times are derived for this class of translators. The upper bound on memory storage is shown to be proportional to the length of the input program. The upper bound on translation time required by these machines is also shown to be proportional to the length of the input program.

A detailed example, drawn from the ALGOL grammar, illustrates an actual application of the ideas in this paper.

Notation and basic definitions

Let V be a finite set of symbols, which we will call the *vocabulary*. Elements of V are denoted by letters, such as d, e, f, G, H, I, etc. Finite sequences of symbols, including the empty sequence e, are called *strings* and are denoted by late small letters, such as x, y, z, etc. The set of all strings over a set such as V is denoted by V^*.

A context-free grammar (abbreviated CFG) is an ordered four-tuple

$$G = (V, T, P, S)$$

where

(a) V is a vocabulary of symbols.
(b) T is a proper subset of V called the *terminals*.
(c) P is a finite, nonempty set of syntactic rules P_i of the form $U \rightarrow x$, where $U \neq x$, U is in $V - T$, and x is in $V^* - \{e\}$. For a rule $P_i = U \rightarrow x$, U is called the *left part* and x the *right part* of P_i.
(d) S is a special symbol in $V - T$, the *initial symbol.*

As is usual, we say that x *directly produces* y. ($x => y$), and conversely y *directly reduces* to x if and only if there exist strings u, v, such that $x = uZv$ and $y = uwv$ and

$$Z \rightarrow w \text{ is in } P .$$

x *produces* y ($x \dot{=}> y$), and conversely y *reduces* to x if and only if either

$$x = y$$

or there exists a sequence of nonempty strings (w_0, $w_1, \ldots, w_n$) such that $x = w_0$ and $y = w_n$ and

$$w_i => w_{i+1} \; (i = 0, 1, \ldots n - 1 \text{ and } n \geq 1) .$$

x is a *sentence* of G if x is in $T^* - \{e\}$ and S produces x.

A *context-free language* (abbreviated CFL) is then the set of terminal strings that can be produced by grammar G from its initial symbol S:

$$L(G) = \{x: (S \dot{=}> x) \;\&\; (x \in T^* - \{e\})\}$$

Let S produce x. A *parse* of the string x into the symbol S is a sequence of rules $P_1, \ldots P_n$ such that P_j directly reduces w_{j-1} into $w_j (j = 1, \ldots, n)$ and $x = w_0$, $S = w_n$.

Let $x = a_1, \ldots a_r$ be a string of symbols a_i in T. Then, in some reduction sequence in which $x = w_0$, let x reduce to $w_j = ua_k \ldots a_r$, with u in V* and $1 \leq k \leq r$. If P_j directly reduces string w_j into w_{j+1} and P_{j1} directly reduces w_{j+1} into w_{j+2}, then (P_j, P_{j1}) is called a *leftmost reduction sequence* if

$$w_{j+1} = u'a_{k'} \ldots a_r \qquad u' \text{ in } V^* \times (V - T)$$

and

$$w_{j+2} = u''a_{k''} \ldots a_r \qquad u'' \text{ in } V^* \times (V - T)$$

and

$$k \leq k' \leq k'' \leq r .$$

or

$$w_{j+1} = u'w \text{ and } w_{j+2} = u'A$$

and

$$A \rightarrow w \epsilon P.$$

A parse $(P_1, \ldots, P_n)$ is called a *leftmost parse* if and only if the sequences (P_i, P_{i+1}) are leftmost reduction sequences for $i = 1, \ldots, n - 1$.

If $(P_1, \ldots, P_n)$ is a parse of string x into symbol S, there exists a permutation of $(P_1, \ldots, P_n)$ that is leftmost. We define an *unambiguous* grammar G to be one in which every x in L(G) has exactly one leftmost parse.

We next define a normal form for CFG's, in terms of which a leftmost parsing algorithm can be designed. The correspondence between this leftmost parsing algorithm and a pushdown automaton model to be introduced will then become apparent. Subsequently, an algorithm for facilitating single-scan leftmost parsing in a large class of grammars will be developed.

Normal form grammars

A grammar G = (V, T, P, S) will be said to be in *normal form* if all the rules in P are of the forms

$$A_i \rightarrow A_{i1}A_{i2} \quad \text{or} \quad A_j \rightarrow A_{j1}$$

or

$$A_k \rightarrow A_{k1}a_{k2} \quad \text{or} \quad A_m \rightarrow a_{m1}$$

with A_{i1}, A_{i2}, A_{j1}, A_{k1} in V − T and a_{k2}, a_{m1} in T. A very simple algorithm exists for converting any grammar H into a grammar H′ in normal form such that L(H) = L(H′). Because of this algorithm, all derivations of sentences in L(H) are in one-to-one correspondence with derivations of sentences in L(H′). The algorithm works as follows:

All productions in P of H that are already in normal form are taken into P′ of H′. The remaining productions in P are of the form

$$X \rightarrow X_1 \ldots X_n \quad , \quad (n > 2) \ \& \ (X_i \in V) .$$

Each production of this form is transferred to P′ as a sequence of productions.

$$J_v \rightarrow J_{v-1}X_{v+1} \quad \text{for} \quad v = 1, \ldots, n - 1$$

where J_{n-1} is X. J_0 is X_1 if X_1 is in V − T of H; otherwise an additional rule of the form

$$J_0 \rightarrow X_1$$

is included in P′. The J_v are treated as new elements in V′ − T′ of H′, and the J_v are distinct from the elements in V − T of H.

The fact that the J_v of the algorithm are "new and distinct" leads to a simple proof of the one-to-one correspondence between derivations of sentences in L(H) and L(H′): Since each rule of P corresponds to a particular rule or sequence of rules in P′, it follows that, for each derivation possible in H, there is a corresponding derivation in H′, and conversely. Because of this unique correspondence, it also follows that ambiguity in L(H) is equivalent to ambiguity in L(H′).

Leftmost parses and normal-form grammars

In order to describe the algorithm for producing leftmost parses of the sentences of a grammar G in normal form, we introduce boundary markers # to the vocabulary of G. A new initial symbol S′ now takes the place of S in G, and three new rules are added to G:

$$P_1' = S' \rightarrow J_1 \ \#$$

$$P_2' = J_1 \rightarrow J_2S \qquad P_3' = J_2 \rightarrow \#$$

This has the effect of putting boundary markers at both ends of all strings produced by the grammar.

Let $w_0 = \# a_1 \ldots a_n \#$ be a string in the language of such a grammar. In the initial step of the leftmost parsing algorithm, rule P_3' is applied, yielding string

$$w_1 = J_2 a_1 \ldots a_n \#$$

After j steps, w_0 has been reduced to

$$w_j = J_2 K_1 \ldots K_r a_s \ldots a_n \# \, (1 \leq r < s \leq n + 1) \, .$$

In this configuration, $K_1, \ldots, K_r$ are all symbols of $V - T$ in the grammar. If w_0 is in L(G), the leftmost sequence of rules $P_3' = P_1, \ldots, P_j$ are precisely the first j reductions of the leftmost parse of w_0 to S'. Capital letters are assumed to be members of $V - T$ of the grammars in the remaining discussion.

For the $(j + 1) - $th reduction, five different cases must be distinguished:

(0) S' does not produce w_j, where $w_j = J_2 K_1 \ldots K_r a_s \ldots a_n \#$.

If S' does produce w_j, we have to distinguish between the following possibilities:

(1) A rule of the form $P_{j+1} = K_{r+1}' \rightarrow a_s$ reduces w_j to w_{j+1}.
(2) A rule of the form $P_{j+1} = K_r' \rightarrow K_r$ reduces w_j to w_{j+1}.
(3) A rule of the form $P_{j+1} = K_{r-1}' \rightarrow K_{r-1}K_r$ reduces w_j to w_{j+1}.
(4) A rule of the form $P_{j+1} = K_r' \rightarrow K_r a_s$ reduces w_j to w_{j+1}.

That only these cases need be considered is proved in [13]. Note that each of the applications of rules P_{j+1} in cases (1)–(4) leaves the length of string w_{j+1} either the same as that of w_j or reduced in length by one symbol. This fact is used in a later proof of the upper bound on computation time required by the leftmost parsing algorithm.

In general, the decision concerning which of the cases (1) to (4) apply for the $(j + 1)$—th step of a leftmost parse must be made in terms of *context*. As an example, there may exist rules in the grammar having $K_{r-1}K_r$ *and* $K_r a_s$ on the right part. To decide which case applies at a given step of the parse then requires algorithms for discovering what symbols can legally be adjacent to the symbols being reduced in that step of the reduction while w_0 is a sentence of G. The algorithms to be given in what follows are similar to those of Floyd[2] and Wirth and Weber[16] in that they construct all legal cooccurrences of triples of symbols in some language from that language's grammar.

Case (1)

For a rule of the form $P_{j+1} = K_{r+1}' \rightarrow a_s$ to apply for the $(j + 1)$—th reduction, there must be one or more symbols Z in $V - T$ such that $Z \rightarrow K_r Y$ is in P and $Y \overset{*}{=}> K_{r+1}'u$, with u in V^*.
Since the set $\{K: W \overset{*}{=}> Ku \;\&\; u \in V^*\}$ can be constructed, the context in which Case (1) applies can be found. The pairs (K_r, a_s) are the contexts in which the rule $K_{r+1}' \rightarrow a_s$ applies.

Case (2)

For a rule of the form $P_{j+1} = K_r' \rightarrow K_r$ to apply for the $(j + 1) - $th reduction, there must be one or more symbols Z in $V - T$ such that either

	$Z \rightarrow K_{r-1}Y$	is in P
and	$Y \overset{*}{=}> Xu$	with u in V^*.
and	$X \rightarrow RT$	where $R \overset{*}{=}> K_r'$
and	$T \overset{*}{=}> a_s y$,	with y in V^*.
or	$Z \rightarrow Ya_s$	is in P
and	$Y \overset{*}{=}> uX$	with u in V^*
and	$X \rightarrow K_{r-1}R$	
where	$R \overset{*}{=}> K_r'$.	

The pairs (K_{r-1}, a_s) are the contexts in which the rule $K_r' \rightarrow K_r$ applies.

Case (3)

For a rule of the form $P_{j+1} = K_{r-1}' \rightarrow K_{r-1}K_r$ to apply for the $(j + 1)$—th reduction, there must be one or more symbols Z in $V - T$ such that

	$Z \rightarrow YX$ is in P
and	$X \overset{*}{=}> a_s u$, with u in V^*.
and	$Y \overset{*}{=}> wK_{r-1}'$ with w in V^*

The pairs (K_{r-1}, a_s) are the contexts in which the rule $K_{r-1}' \rightarrow K_{r-1}K_r$ applies.

Case (4)

For a rule of the form $P_{j+1} = K_r' \rightarrow K_r a_s$ to apply for the (j + 1) − th reduction, there must be one or more symbols Z in V − T such that

$$Z \rightarrow K_{r-1}Y \text{ is in P}$$

and $$Y \overset{+}{=} > K'_r u \text{ with u in } V^*.$$

The pairs (K_{r-1}, a_s) are the contexts in which rule $K'_r \rightarrow K_r a_s$ applies.

After the contexts for which cases (1)–(4) apply have been determined, there may in general still exist rules having the same contexts. The existence of such rules in a grammar may imply the necessity of backtracking methods for use in parsing a given string of that grammar. Or, such a grammar may be ambiguous. In the following section, we sketch a formal model for this normal-form leftmost parsing algorithm. In terms of this model, we can present an algorithm for eliminating the necessity of backtracking in a large class of unambiguous CFL's.

Pushdown automaton parsing model

In this section, we present an automaton model of our leftmost parsing algorithm. This model will be called a *bounded-context acceptor* (BCA), and can be described intuitively as follows: It has a finite number of states, a finite input vocabulary, and an auxiliary memory stack mechanism (called a *pushdown store*). In addition, it has an initial (or starting) state, a final (or accepting) state, and a boundary symbol ⋕ whose function in the scheme is analogous to the control cards used before and after a computer program. In general, the BCA is in some state, and is scanning the topmost pushdown-store symbol *K* and an input-string symbol *a*. In a possibly nondeterministic manner, the BCA goes to another state, either by erasing *K* or by erasing *a* in the process, and possibly storing a new symbol on top of the pushdown-store. All these notions are restated more precisely in what follows.

A *bounded-context acceptor* P is defined to be a seven-tuple:

$$P = (Q, T, N, M, ⋕, S_0, F),$$

where

a. Q is a finite set, called the *states* of the machine.
b. T is a finite set of symbols, called the *input-tape vocabulary*.
c. N is a finite set of symbols, called the *pushdown-store vocabulary*.
d. M is a mapping of NxQxT into the finite subsets of $N x N x \{S_0\} x (T \cup \{e\}) \cup (N \cup \{e\}) x Q x (T \cup \{e\}) - \{e\} x Q x \{e\}$.
e. S_0 is the *initial state* of the computation, and F is called the *final state*.
f. ⋕ is a special symbol such that

$$T \cap N = \{⋕\}.$$

Let A = (Q, T, N, M, ⋕, S_0, F) be a BCA. A *configuration* of A is an element of

$$\{⋕\} x (N - \{⋕\})^* x Q x (T - \{⋕\})^* x \{⋕\}.$$

the configuration (⋕ t S_1 y ⋕) denotes the fact that the acceptor A is in state S_1, with string ⋕t on the pushdown store and string y⋕ remaining to be read on the input tape.

By analogy to our notation for CFL's, we can define an *initial configuration* of a computation to be

$$C_0 = (⋕\ S_0\, x\ ⋕),$$

where $x \in T^* - \{e\}$ is the input string to be accepted. The *final configuration* is (⋕ F ⋕). The computation performed by a BCA is essentially a reduction sequence that reduces C_0 to the final configuration.

Let C_j and C_{j+1} be two configurations of a computation. Then, C_j *direztly reduces* to C_{j+1}, or $C_j \mid\!- C_{j+1}$, if

$$C_j = (t\ Z\ S_1\ a\ w),\ C_{j+1} = (t\ y\ S_2\ b\ w)$$

and where

$$(y, S_2, b) \in M(Z, S_1, a)$$

and

$$[(t = e) \& (Z = ⋕) \vee (t \in \{⋕\} x (N - \{⋕\})^*) \& (Z \in (N - \{⋕\}))]$$

$$\& [(w = e) \& (a = ⋕) \vee (w \in (T - \{⋕\})^* x \{⋕\}) \& (a \in (T - \{⋕\}))]$$

$$\& \sim [(Z = ⋕) \& (a = ⋕)]$$

and

$$(b \in (\{a\} \cup \{e\})) \& [(y \in (Z \cup \{e\}) x (N \cup \{e\})) \& (Z \neq ⋕) \vee (y \in \{⋕\} x (N \cup \{e\})) \& (Z = ⋕)].$$

We next define the sequence of configurations that leads to a complete computation of the acceptor.

Let

$$C_1 = (\#uS_1a_1 \ldots a_kv\#)\ ,\ u \in N^* \text{ and } v \in T^*$$

and

$$C_2 = (\#u'S_{j+1}v\#)\ ,\ u' \in N^*\ ,$$

be configurations of the computation. Then C_1 *reduces* to C_2, or $C_1 \overset{*}{\vdash} C_2$, if there exists a sequence of configurations $(H_0, H_1, \ldots, H_j)$, with $C_1 = H_0$ and $C_2 = H_j$ and

$$H_{i-1} \vdash H_i \qquad (i = 1, \ldots, j)\ .$$

Then, the *language* accepted by a BCA P is the set of input strings given by

$$L(P) = \{x: [(\#\ S_0x\#) \overset{*}{\vdash} (\#\ F\ \#)]\ \&\ [x \in T^* - \{e\}]\}\ .$$

In the more standard pushdown-automaton acceptor model,[3] M is treated as a mapping from

$$(N - \{\#\})\ x\ Q\ x\ (T\ U\ \{e\}\ -\ \{\#\})$$

into the finite subsets of $(N - \{\#\})^*\ x\ Q$. This formalism implies the existence of transitions between states in which the current input-tape symbol is ignored (i.e., the empty symbol e is erased from the input tape). When an input-tape symbol is used for a transition in this model, it is always erased during the transition. Thus, while a standard pushdown automaton having the same language can always be constructed from a BCA, the notion of context in determining a transition is lost, and the resulting acceptor will generally have fewer uniquely defined transitions than the original BCA. Moreover, given a standard pushdown automaton, a grammar for its language can always be found,[3] and, from this grammar, a BCA can be constructed to accept the same language, as will be seen. Hence, BCA's and standard pushdown automata are equivalent in computational power, but a BCA with uniquely defined transitions does not always correspond to a pushdown automaton with uniquely defined transitions.

Automaton realization of leftmost parses

With the BCA model defined above, it is possible to introduce a correspondence between rules of a normal-form grammar and the states and symbols of a BCA. In this correspondence, the initial symbol S of a grammar becomes the final state F of the BCA. Furthermore, the BCA constructed from a normal-form grammar is a slightly restricted version of the BCA model. This is because transitions from initial state S_0 of a constructed BCA can only occur together with the erasure of an input-tape symbol. Since the full BCA model only allows the pushdown store to be increased in size during a transition into state S_0, this additional restriction means that the pushdown store of a constructed BCA can increase in size by at most one symbol for each input-tape symbol read during a computation.

The BCA is constructed from a normal-form grammar as follows: For all rules in the grammar of the forms

$$A_i \rightarrow A_{i1}A_{i2} \quad \text{and} \quad A_j \rightarrow A_{j1}a_{j2}$$

(where the capital letters are nonterminals and the small letters are terminal symbols), the A_{j1}'s and A_{i2}'s become states of the BCA. The A_{i1}'s become members of N, the stack vocabulary, and the a_{j2}'s become members of T, the input-string vocabulary. What follows is the algorithm for constructing a BCA that accepts the language of a normal-form grammar. The algorithm is in four sections corresponding to the four rule types allowed in normal-form grammars.

I. Rule $A_i \rightarrow A_{i1}A_{i2}$ with contexts (A_{i1}, a_s):

If $A_i \in N$, then $(A_i, S_0, a_s) \in M(A_{i1}, A_{i2}, a_s)$.

If $A_i \in Q$, then $(e, A_i, a_s) \in M(A_{i1}, A_{i2}, a_s)$.

These transitions take care of all possibilities arising from case (3) of the leftmost parsing algorithm. If A_i is in N, that means that a pair of nonterminals A_iA_{j2} appears on the right part of some rule of the grammar. Hence, A_i is stored on top of the stack, and the automaton transfers to its initial state, from which it proceeds to discover A_{j2}. If A_i is in Q, then A_i is either the second nonterminal of some rule of the grammar or is the first nonterminal in a rule of the form $A_k \rightarrow A_ia_{k2}$. In either case, the stack does not increase in length during the transition.

II. Rule $A_k \rightarrow A_{k1}a_{k2}$ with contexts (K_{r-1}, a_{k2}):

If $A_k \in N$, then $(K_{r-1}A_k, S_0, e) \in M(K_{r-1}, A_{k1}, a_{k2})$.

If $A_k \in Q$, then $(K_{r-1}, A_k, e) \in M(K_{r-1}, A_{k1}, a_{k2})$.

These transitions take care of all possibilities arising from case (4) of the leftmost parsing algorithm.

III. Rule $A_j \rightarrow a_s$ with contexts (K_r, a_s):

If $A_j \in N$, then $(K_rA_j, S_0, e) \in M(K_r, S_0, a_s)$.

If $A_j \in Q$, then $(K_r, A_j, e) \in M(K_r, S_0, a_s)$.

These transitions take care of all possibilities arising from case (1) of the leftmost parsing algorithm.

IV. Rule $A_j \rightarrow A_{j1}$ with contexts (K_{r-1}, a_s):

For every chain of rules in the grammar of the form

$$P_1 = A \rightarrow A^{(1)}$$
$$P_2 = A^{(1)} \rightarrow A^{(2)}, \ldots,$$
$$P_n = A^{(n-1)} \rightarrow A^{(n)} \quad \text{with } n \geq 2,$$

and such that there is at least one context (K'_{r-1}, a'_s) common to rules $(P_1, \ldots, P_n)$, we introduce automaton transitions of the form

$$(K'_{r-1}, A^{(n-1)}, a'_s) \in M(K'_{r-1}, A^{(n)}, a'_s)$$
$$\vdots$$
$$(K'_{r-1}, A^{(1)}, a'_s) \in M(K'_{r-1}, A^{(2)}, a'_s).$$

These $(A^{(1)}, \ldots, A^{(n)})$ are thus treated as states of the BCA.

For all contexts (K_{r-1}, a_s) associated with individual rules $A_j \rightarrow A_{j1}$, we have the following:

If $A_j \in N$, then $(K_{r-1}A_j, S_0, a_s) \in M(K_{r-1}, A_{j1}, a_s)$.
If $A_j \in Q$, then $(K_{r-1}, A_j, a_s) \in M(K_{r-1}, A_{j1}, a_s)$.

These transitions take care of all possibilities arising from case (2) of the leftmost parsing algorithm.

When all transitions of a machine have been defined as described above, the language accepted by that machine is the language of the grammar from which it is constructed.[14]

A simple programming language translator

The following is a simplified grammar for a computer programming language having nested block structure, conditional statements, and arithmetic assignment statements. The ALGOL conventions are used for representing symbols of the grammar; i.e., members of V − T are enclosed by the metasyntactic brackets "⟨" and "⟩", and members of T are not. The symbol "|" is a separator that allows two or more rules having the same left part to be written together.

Table I—A grammar for a simple programming language

```
G:  ⟨program⟩ → ⟨body⟩ ⟨stat⟩ end
    ⟨body⟩ → begin | ⟨body⟩ ⟨stat⟩;
    ⟨stat⟩ → ⟨program⟩ | ⟨assignment⟩
    ⟨assignment⟩ → ⟨var⟩ := ⟨expr⟩
    ⟨expr⟩ → ⟨simple expr⟩ | ⟨if clause⟩
        ⟨simple expr⟩ else ⟨expr⟩
    ⟨simple expr⟩ → ⟨term⟩ | ⟨simple expr⟩
        + ⟨term⟩
    ⟨term⟩ → ⟨factor⟩ | ⟨term⟩ * ⟨factor⟩
    ⟨factor⟩ → ⟨var⟩ | ⟨number⟩ | [⟨expr⟩]
    ⟨if clause⟩ → if ⟨relation⟩ then
        ⟨relation⟩ → ⟨simple expr⟩ = ⟨simple expr⟩
    ⟨var⟩ → A|B|C| . . . |Z
    ⟨number⟩ → ⟨digit⟩ | ⟨number⟩ ⟨digit⟩
    ⟨digit⟩ → 0|1| . . . |9
```

The programming language G easily reduces to the following normal-form grammar G′ augmented by the addition of endmarkers:

Table II—The normal-form version of grammar G

```
G′: S → Y1#

  Y1 → Y2 ⟨program⟩                      ⟨factor⟩ → ⟨var⟩ | ⟨number⟩ | X8]
  Y2 → #                                    X8 → X9 ⟨expr⟩
  ⟨program⟩ → X1 end                        X9 → [
    X1 → ⟨body⟩ ⟨stat⟩                      ⟨if clause⟩ → X10 then
  ⟨body⟩ → begin | X2;                      X10 → X11 ⟨relation⟩
    X2 → ⟨body⟩ ⟨stat⟩                      X11 → if
  ⟨stat⟩ → ⟨program⟩ | ⟨assignment⟩         ⟨relation⟩ → X12 ⟨simple expr⟩
  ⟨assignment⟩ → X3 ⟨expr⟩                  X12 → ⟨simple expr⟩ =
    X3 → ⟨var⟩ :=                           ⟨var⟩ → A|B|C| . . . |Z
  ⟨expr⟩ → ⟨simple expr⟩ | X4 ⟨expr⟩        ⟨number⟩ → ⟨digit⟩ | ⟨number⟩ ⟨digit⟩
    X4 → X5 else                            ⟨digit⟩ → 0|1| . . . |9
    X5 → ⟨if clause⟩ ⟨simple expr⟩
  ⟨simple expr⟩ → ⟨term⟩ | X6 ⟨term⟩
    X6 → ⟨simple expr⟩ +
  ⟨term⟩ → ⟨factor⟩ | X7 ⟨factor⟩
    X7 → ⟨term⟩*
```

What follows is a table of contexts in which the rules of G′ can be applied during a leftmost parse of some string in L(G′):

Table III—Contexts associated with rules in Table II

Rule	*Contexts*
S → Y_1 ⋕	(e, ⋕)
Y_1 → Y_2 ⟨program⟩	(Y_2, ⋕)
Y_2 → ⋕	(e, ⋕),
⟨program⟩ → X_1 *end*	(Y_2, *end*), (⟨body⟩, *end*)
X_1 → ⟨body⟩ ⟨stat⟩	(⟨body⟩, *end*)
⟨body⟩ → *begin*	(Y_2, *begin*), (⟨body⟩, *begin*)
⟨body⟩ → X_2;	(Y_2, ;), (⟨body⟩, ;)
X_2 → ⟨body⟩ ⟨stat⟩	(⟨body⟩, ;)
⟨stat⟩ → ⟨program⟩	(⟨body⟩, *end*), (⟨body⟩, ;)
⟨stat⟩ → ⟨assignment⟩	(⟨body⟩, *end*), (⟨body⟩, ;)
⟨assignment⟩ → X_3 ⟨expr⟩	(X_3, *end*), (X_3, ;)
X_3 → ⟨var⟩ :=	(⟨body⟩, :=)
⟨expr⟩ → ⟨simple expr⟩	(X_3, ;), (X_3, *end*), (X_9,])
⟨expr⟩ → X_4 ⟨expr⟩	(X_4, *end*), (X_4, ;)
X_4 → X_5 *else*	(X_3, *else*), (X_9, *else*)
X_5 → ⟨if clause⟩ ⟨simple expr⟩	(⟨if clause⟩, *else*)
⟨simple expr⟩ → ⟨term⟩	(⟨if clause⟩, *else*), (X_3, ;), (X_3, *end*, (X_9,]), (⟨if clause⟩, +), (X_4, +), (X_3, +), (X_9, +), (X_{11}, =)
⟨simple expr⟩ → X_6 ⟨term⟩	(x_6, +), (X_6, *else*), (X_6, ;), (X_6, *end*), (X_6, =), (X_6, *then*), (X_6,])
X_6 → ⟨simple expr⟩ +	(⟨if clause⟩, +), (X_4, +), (X_3, +), (X_9, +), (X_{12}, +), (X_{11}, +)
⟨term⟩ → ⟨factor⟩	(X_6, +), (X_6, =), (X_6, *then*), (X_6, *else*) (X_6, *), (⟨if clause⟩, *), (X_{12}, *), (X_4, *), (X_9, *), (X_3, *), (⟨if clause⟩, *else*), (X_3, ;), (X_3, *end*), (X_9,]), (⟨if clause⟩, +), (X_4, +), (X_3, +), (X_9, +), (X_{11}, =)

Rule	*Contexts*
⟨term⟩ → X_7 ⟨factor⟩	(X_7, +), (X_7, =), (X_7, *then*), (X_7, *else*) (X_7,]), (X_7, ;), (X_7, *end*)
X_7 → ⟨term⟩*	(X_6, *), (⟨if clause⟩, *), (X_{12}, *), (X_{11}, *) (X_4, *), (X_3, *), (X_9, *)
⟨factor⟩ → ⟨var⟩	(X_7, *), (X_7, +), (X_7, =), (X_7, *then*),
⟨factor⟩ → ⟨number⟩	(X_7, *else*), (X_6, +), (X_6, =), (X_6, *then*), (X_6, *else*), (X_6, *else*), (X_6, *), (⟨if clause⟩, *), (X_{12}, *), (X_4, *), (X_9, *), (X_3, *), (⟨if clause⟩, *else*), (X_3, ;), (X_3, *end*), (X_9,]), (⟨if clause⟩, +), (X_4, +), (X_3, +), (X_9, +), (S_{11}, =)
⟨factor⟩ → X_8]	(X_7,]), (X_6,]), (⟨if clause⟩,]), (X_{12},]) (X_{11},]), (X_4,]), (X_9,]), X_3,])
X_8 → X_9 ⟨expr⟩	(X_9,])
X_9 → [	(X_9, [), (X_7, [), (X_6, [), (⟨if clause⟩, [) (X_{12}, [), (X_{11}, [), (X_3, [), (X_4, [)
⟨if clause⟩ → X_{10} *then*	(X_3, *then*), (X_4, *then*), (X_9, *then*)
X_{10} → X_{11} ⟨relation⟩	(X_{11}, *then*)
X_{11} → *if*	(X_3, *if*), (X_4, *if*), (X_9, *if*)
⟨relation⟩ → X_{12} ⟨simple expr⟩	(X_{12}, *then*)
X_{12} → ⟨simple expr⟩ =	(X_{11}, =)
⟨var⟩ → A	(X_7, A), (⟨body⟩, A), (X_6, A), (X_4 A), (X_9, A), (X_3, A), (X_{12}, A), (X_{11}, A), (⟨if clause⟩, A)
⟨number⟩ → ⟨number⟩ ⟨digit⟩	(⟨number⟩, 1), . . ., (⟨number⟩, 9) (⟨number⟩, *), (⟨number⟩, +) (⟨number⟩, =), (⟨number⟩, *then*) (⟨number⟩, *else*), (⟨number⟩, ;) (⟨number⟩, *end*)
⟨digit⟩ → $0 \mid 1 \mid \ldots \mid 9$	(⟨number⟩, 0), . . ., (⟨number⟩, 9), (⟨if clause⟩, 0), . . ., (⟨if clause⟩, 9), (X_7, 0), . . ., (X_7, 9), (X_6, 0), . . ., (X_6, 9), (X_4, 0), . . ., (X_4, 9), (X_{11}, 0), . . ., (X_{11}, 9), (X_{12}, 0), . . ., (X_{12}, 9),

From Table III, a flow chart of the BCA that accepts L(G′) can be constructed. This flow chart is abbreviated in that, for a given state, only those contexts necessary for determining a transition are presented. Thus, when no ambiguity will be introduced, only the stack symbol or the input-string symbol is used for determining which of several possible transitions can occur. In the flow chart, the array N, with index i, is used to represent the symbols of the stack, and the array S, with index j, represents the symbols of the input string. Note that "error exits," i.e., the instances of case (0) in the leftmost parsing algorithm, are omitted from the flow chart. An error is assumed to exist at any transition in which the appropriate symbols are not present on either the input string or the stack.

The next step after synthesizing a BCA as in Figure 1 is to design a translator for the language accepted by that BCA. To do this, we employ a notation similar to that used in [13], and available in numerous versions in the current literature. The basic idea of the notation is to introduce *rules of translation* in a one-to-one correspondence with the rules of the original grammar. These rules of translation describe the effect of translating the right parts of the syntactic rules with which they are associated. As an example, we might have the following pairing in some grammar:

Syntactic Rule:	*Rule of Translation:*
⟨term⟩ → ⟨term⟩ * ⟨factor⟩	⟨term⟩ ⟨factor⟩ *multiply*

This pairing of rules can be represented as a *translation grammar* $G^t = (G, 0, f)$, where $G = (V, T, P, S)$ is the programming language syntax, 0 is a translated program vocabulary, and f is a one-to-one mapping from P into Px0*. The rule of translation given in the above example is easily recognized as one rule for converting from standard arithmetic notation to reverse Polish notation. In the translated sequence '⟨term⟩ ⟨factor⟩ *multiply*', the translated objects corresponding to ⟨term⟩ are written out in the sequence determined by the rules of translation associated with the syntactic rules derived from ⟨term⟩, and likewise with ⟨factor⟩. In general, if a rule of translation is identical to the right part of its associated syntax rule, we write the symbol 'I' in place of the rule of translation.

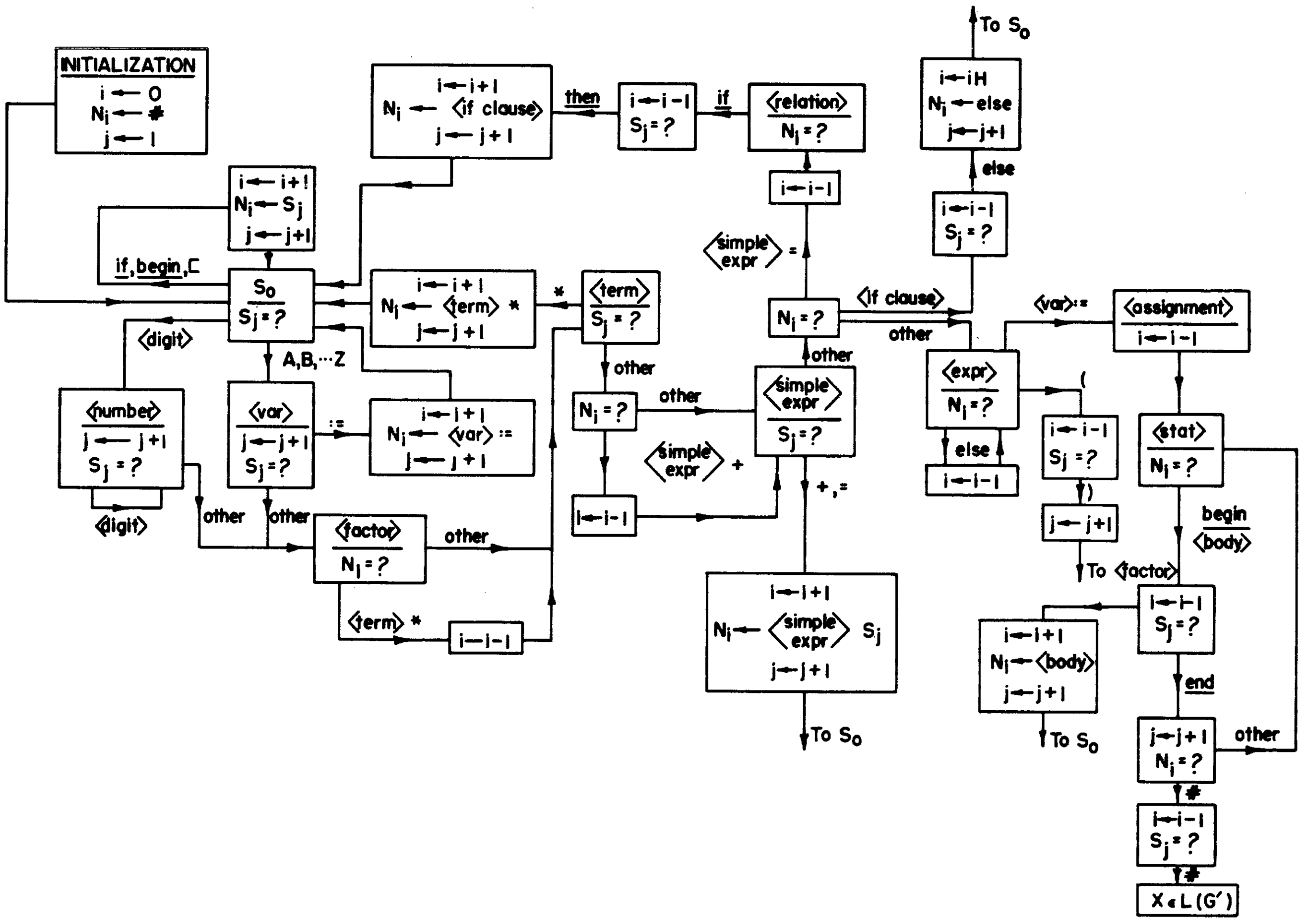


Figure 1—The BCA acceptor of L(G′)

We can next present the simple programming language given above as a translation grammar:

Table IV—Translation grammar for a small language

Syntactic Rules:	*Rules of Translation:*
G: ⟨program⟩ → ⟨body⟩ ⟨stat⟩ *end*	I
⟨body⟩ → *begin*	I
⟨body⟩ → ⟨body⟩ ⟨stat⟩;	I
⟨stat⟩ → ⟨program⟩	I
⟨stat⟩ → ⟨assignment⟩	I
⟨assignment⟩ → ⟨var⟩ := ⟨expr⟩	⟨var⟩ ⟨expr⟩ *assign*
⟨expr⟩ → ⟨simple expr⟩	I
⟨expr⟩ → ⟨if clause⟩ ⟨simple expr⟩ *else* ⟨expr⟩	⟨if clause⟩ ⟨simple expr⟩ *then* ⟨expr⟩ *else*
⟨simple expr⟩ → ⟨term⟩ ⟨simple expr⟩ → ⟨simple expr⟩ + ⟨term⟩	I ⟨simple expr⟩ ⟨term⟩ *add*
⟨term⟩ → ⟨factor⟩	I
⟨term⟩ → ⟨term⟩ * ⟨factor⟩	⟨term⟩ ⟨factor⟩ *multiply*
⟨factor⟩ → ⟨var⟩	I
⟨factor⟩ → ⟨number⟩	*numberoperand* ⟨number⟩
⟨factor⟩ → [⟨expr⟩]	⟨expr⟩
⟨if clause⟩ → *if* ⟨relation⟩ *then*	⟨relation⟩ *if*
⟨relation⟩ → ⟨simple expr⟩$^{(1)}$ = ⟨simple expr⟩$^{(2)}$	⟨simple expr⟩$^{(1)}$ ⟨simple expr⟩$^{(2)}$ *equals*
⟨var⟩ → A ⋮ ⟨var⟩ → Z	*variableoperand* A ⋮ *variableoperand* Z
⟨number⟩ → ⟨digit⟩	I
⟨number⟩ → ⟨number⟩ ⟨digit⟩	10 ⟨number⟩ *multiply* ⟨digit⟩ *add*
⟨digit⟩ → 0 ⋮ ⟨digit⟩ → 9	I ⋮ I

The details of the translator grammar can be explained briefly: Essentially, arithmetic expressions and relations are translated into reverse-Polish strings through the rules of translation. Conditional expressions are rearranged so that *if*, *then*, and *else* become place-markers in the translated program, and the device that interprets the translated program contains routines for passing to the statements directly following *if*, *then*, or *else* as appropriate. Since the effect of interpreting the translated program is to coalesce assignment statements into a single resultant operand that is the "value" of the assigned expression, the semicolon ";" that separates program statements is written into the translated program so that the interpreting mechanism can erase the resultant operand of an assignment. ***begin*** and ***end*** are likewise written in sequence into the translated program so that the interpretor of this program can maintain a list of valid identifiers corresponding to the program's nested block structure.

The translator of Figure 2 is thus a relatively straightforward extension of the PDA in Figure 1, with the additional structure arising from the appropriate rules of translation. The sequencing of operators to follow pairs of operands is accomplished by noting that state transitions such as the one that recognizes the sequence

⟨simple expr⟩ + ⟨term⟩

in Figure 1 are appropriate points for writing out operators (here, "*add*") into the translated program. Likewise, a rule of translation such as

numberoperand ⟨number⟩

requires some temporary storage in the translator to store the symbols that comprise ⟨number⟩, finally writing out the translated sequence

Code ('numberoperand')

Code (temporarystore).

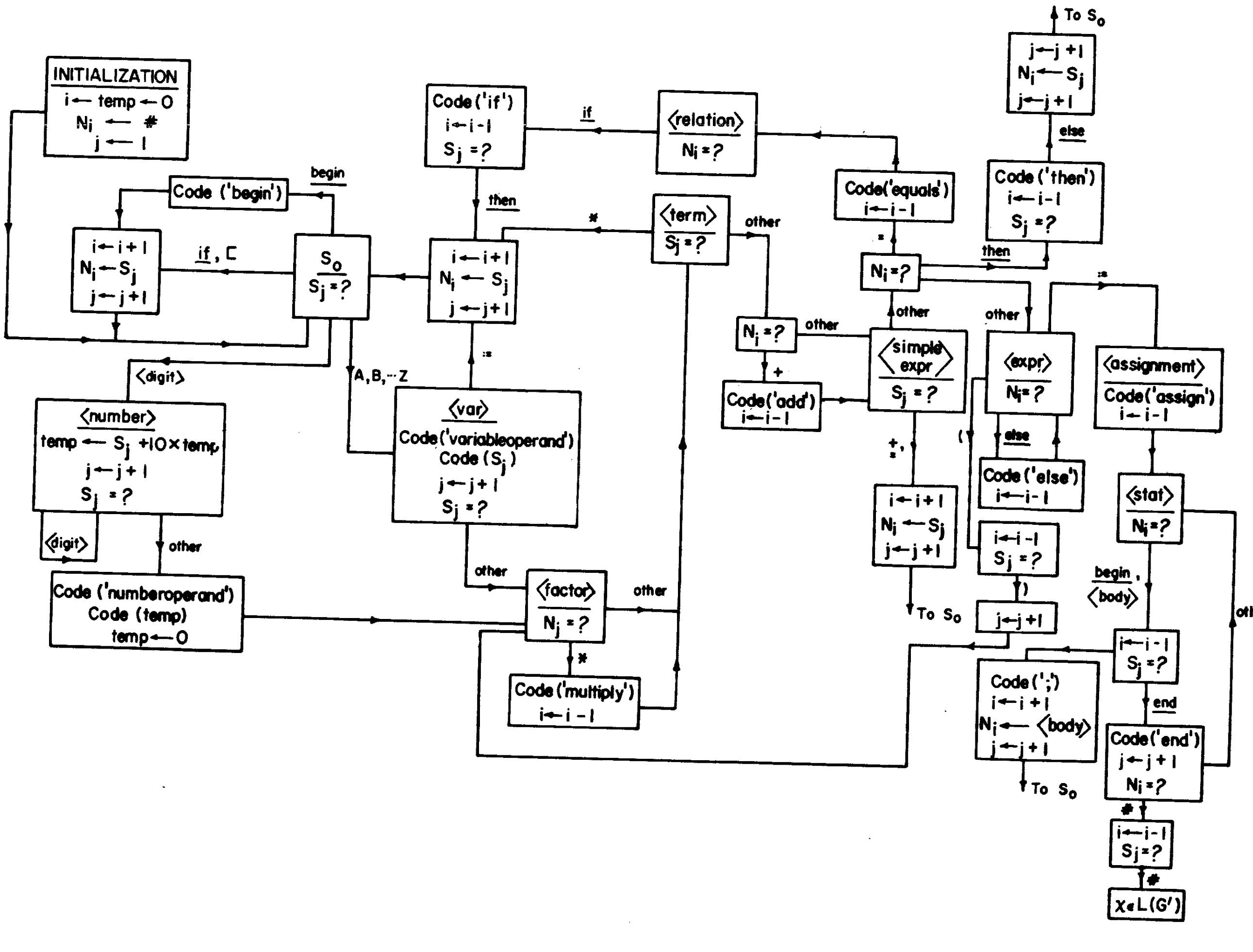


Figure 2—The BCA translator of L(G′)

Deterministic and extended-deterministic acceptors

An acceptor A is called *deterministic* if M is a partial mapping from

$$NxQxT \text{ into } NxNx\{S_0\}x(T \cup \{e\}) \cup (N \cup \{e\}) \, x \, Q \, x \, (T \cup \{e\}) - \{e\} \, x \, Q \, x \, \{e\}.$$

This is equivalent to saying that, for every configuration C_1 of A, there is at most one configuration C_2 for which $C_1 \vdash C_2$. By an induction argument, if x is in L(A), there is only one sequence of configurations by which $(\#S_0x\#) \overset{*}{\vdash} (\#F\#)$. Thus, if A were constructed from some grammar G, there would exist exactly one leftmost parse for each x in L(G), and G would be unambiguous. However, it is fairly easy to construct languages that are unambiguous and for which no deterministic BCA can be constructed.[13]

Since not every unambiguous grammar leads to a deterministic BCA, it is of interest to consider methods for extending the BCA model to handle a larger class of unambiguous languages in some "almost deterministic" fashion. This is important because any nondeterministic automaton is inefficient to use, owing to the necessity of repeating its computations until the correct sequences of configurations are found. This necessity for backtracking during computations of such an automaton A with input string x in L(A) occurs when a configuration C_1 is reached for which

$$C_1 \vdash C_{11}, \ldots, C_1 \vdash C_{1n} \quad \text{and} \quad n \geq 2 .$$

Since we assume that L(A) is unambiguous, there can exist only one C_{1j} above for which $C_{1j} \overset{*}{\vdash} (\#F\#)$. The problem then becomes one of finding a general algorithm for processing each of the n possibilities above, together with their descendents, in some *parallel* fashion, perhaps similar to the methods used for "real-time" languages.[4,11]

The algorithm that we present here for implementing parallel computations of nondeterministic BCA's involves a basic computational strategy: No matter how many alternative configurations exist at some step in a computation, all the configurations must have the same input-string symbol in common. Thus, simulation of a parallel computation having no more than n configurations active requires only one input string and n pushdown stores. Moreover, the number of these stacks in use can shrink and grow during a computation, as possible configurations are rejected or added. As will become apparent, at most p stacks will ever be in use at once, where p is the number of states in the original BCA. In what follows, it will also be apparent that there is no "communication" between the stacks; i.e., the extended BCA that results is still equivalent in computational power to the BCA from which it originated.

Given an initially nondeterministic acceptor A, the algorithm constructs sets of states from A, and extends the transition table M to include transitions between these sets of states. Transitions are also defined between individual states of A and sets of states, and between sets of states and individual states. In these transitions involving sets of states, each state in a set is associated with a single stack, and the number of stacks in operation during a computation increases or decreases depending on the relative sizes of the sets between which transitions occur.

The algorithm for constructing sets of states operates in two phases: In the first phase, sets of states S_{ij} are constructed from individual states S_i of the original acceptor. These constructed sets have the property that, for a given context (K_{r-l}, a_s), each state in S_{ij} corresponds to one of the configurations that can result from a configuration containing state S_i. In the second phase, the sets of states S_{ij} from the first phase are used to construct further sets of states S_{ijk}. These sets have the property that, for a given context (K', a_s), each state in S_{ijk} appears in a configuration derivable from some state in S_{ij}.

There are two cases in each phase of the extension algorithm, corresponding to the erasure or non-erasure of an input-string symbol by the configurations active in some step of a computation. The extension algorithm acts so as to force the state-set transitions of an acceptor to proceed by first erasing as many pushdown-store symbols as can be erased before all states in the set that results are ready to erase an input-string symbol. Hence, on transitions from a set of states in which, for some context, at least one state in the set erases a stack symbol, the remaining states that take part in the transition are "inactive." These inactive states present in a transition are carried along from one transition to the next until a configuration is reached in which all states in that state set can read the input-string symbol simultaneously. The formalism of the extension algorithm in the following section merely implements these ideas, and provides a method of determining whether, for a given BCA, this extension algorithm is adequate to prevent the necessity of backtracking.

Multiple configurations

A *multiple configuration* C′ of some acceptor A is a triple

$$C' = ((\#v_1, \ldots, \#v_m) \, S_c \, w \, \#)$$

where the v_i are in N^*, S_c is in $(P(Q)-Q)$ (P(Q) is the set of subsets of Q), w is in T*, and the number of states in S_s is m.

Given a BCA, let $\vdash$ be the relation on

$$(\#N^*xQxT^*\#) \cup (\#N^*x \ldots x\#N^*x(P(Q) - Q)xT^*\#)$$

defined as follows:

I. Let $C_1 = (vKS_1aw)$,

where $[(v = e) \& (K = \#) \vee (v \in \{\#\}x(N - \{\#\})^*) \& (K \in (N - \{\#\}))]$

$\& [(w = e) \& (a = \#) \vee (w \in (T - \{\#\})^* x \{\#\}) \& (a \in (T - \{\#\}))]$

$\& \sim [(K = \#) \& (a = \#)] \& [S_1 \in Q]$.

If (K, a) is a context of S_1 from which more than one configuration can be derived, then one of the following two types of transitions can be defined using multiple configurations. When cases (a) and (b) below both succeed for the same BCA state S_1 and context (K, a), then this method fails for that BCA.

(a) Let

$$S_c = \{c_i: [(y_i, c_i, a) \in M(K, S_1, a)] \& [y_i \in (\{K\} \cup \{e\})x(N \cup \{e\})]\}$$

Then,

$$C_2' = ((vy_1, \ldots, vy_m)\, S_c\, aw)$$

and

$$C_1 \vdash C_2'.$$

We say that

$$M(K, S_1, a) = ((y_1, \ldots, y_m), S_c, a).$$

(b) Let

$$S_c = \{c_i: [(y_i, c_i, e) \in M(K, S_1, a)] \& [y_i \in \{K\}x(N \cup \{e\})]\}$$

Then,

$$C_2' = ((vy_1, \ldots, vy_m)S_cw)$$

and

$$C_1 \vdash C_2'.$$

We say that

$$M(K, S_1, a) = ((y_1, \ldots, y_m), S_c, e).$$

II. Let $C_1' = ((v_1K_1, \ldots, v_tK_t)S_adw)$,

where $[(v_i = e) \& (K_i = \#) \vee (V_i \in \{\#\} x (N - \{\#\})^*) \& (K_i \in (N - \{\#\}))]$

$\& \quad [(w = e) \& (d = \#) \vee (w \in (T - \{\#\})^* x \{\#\}) \& (d \in (T - \{\#\}))]$

$\& \quad \sim [(K_i = \#) \& (d = \#)] \& [S_a \in P(Q) - Q]$

In reality, C_1' stands for t different BCA configurations, one configuration corresponding to each triple

$$(v_jK_ja_jdw)\text{ , where } j = 1, \ldots, t \text{ and } a_j \in S_a .$$

To discover what configurations can result from C_1' in a computation, it is necessary to trace the descendants of each of the t configurations represented by C_1'. In the process of tracing descendant configurations, our strategy will be to force as many transitions as possible in which pushdown-store symbols are erased by states in the sets constructed. Ultimately, a state or a set of states will result in which, for one or more contexts, that state or state set can erase an input-string symbol. The following construction illustrates this principle:

(a) Let $S_B = \{B_i: (\exists a_j)[(a_j \in S_a) \& [(y_i, B_i, d) \in M(K_i, a_j, d)] \& [y_i \in (\{K_i\} \cup \{e\})x(N \cup \{e\})]\}$

$\cup \{a_k: (a_k \in S_a) \& [[\{K_i\}x(N \cup \{e\})xQ \supseteq M(K_i, a_k, d)]\}$

Then, $C_2' = ((v_{i1}\, y_{i1}, \ldots, v_{ij}y_{ij}, v_{k1}K_{k1}, \ldots, v_{kn}K_{kn})S_Bdw)$

and $C_1' \vdash C_2'$. We say that

$$M((K_{i1}, \ldots, K_{ih}), S_a, d) = ((y_{i1}, \ldots, y_{ij}), S_B, d)$$

(b) Let $S_B = \{B_i: \exists a_j)\ [(a_j \in S_a) \& [(y_i, B_i, e) \in M(K_i, a_j, d)]$

$\& [y_i \in \{K_i\}x(N \cup \{e\})]] \& \sim (\exists a_k)\ [(a_k \in S_a)$

$\& [(\{K_i\} \cup \{e\})x(N \cup \{e\})\, xQx\{d\} \supseteq M(K_i, a_k, d)]\}$

Then, $C_2' = ((v_{i1}y_{i1}, \ldots, v_{ip}y_{ip})S_Bw)$

and $C_1' \vdash C_2'$. We say that

$$M((K_1, \ldots, K_t), S_a, d) = ((y_{i1}, \ldots, y_{ip}), S_B, e)$$

Note here that S_B in one of the transitions above could be a set consisting of one or more states. When S_B is a set consisting of one state, then the transition defined from S to S_B has gotten rid of the alternative configurations represented by S_a and its contexts.

We see that the transitions constructed above from a single configuration to a multiple configuration and from one multiple configuration to another preserve the actions that would be taken by the original BCA. In particular, the one successful reduction sequence of a BCA over a string x is contained in the extended reduction sequence involving multiple configurations.[13] The extra stacks used during a multiple configuration reduction sequence simply keep track of additional possibilities until all but one sequence of configurations is eliminated.

In part I of the extension algorithm, the transitions are not uniquely defined for those BCA's in which, for a state s and context (K, a), the following two conditions apply for the same context (K, a):

$$[(K \cup \{e\}) \times (N \cup \{e\}) \times Q \times \{a\} \supseteq M(K, s, a)]$$
$$\& [\{K\} \times (N \cup \{e\}) \times Q \times \{e\} \supseteq M(K, s, a)]$$

When these conditions occur simultaneously, the necessity of simultaneously erasing and not erasing the same input-string symbol during a transition is incompatible with our parallel-processing algorithm. It may be possible to use "lookahead" techniques to decide which of the two types of transitions above should take place by scanning further symbols of the input tape. The use of these lookahead techniques to improve the extension algorithm will be discussed in another paper.

The remaining condition that leads to difficulties in the algorithm arises when there exists a state g in some multiple configuration S_g, such that g is immediately descended from two or more states $y_{i1}, \ldots y_{ig}$ in some S_y for which

$$M((w_1, \ldots, w_m), S_y, v) = ((y_1, \ldots, y_n), S_g, u) \; .$$

For this transition, $\vdash$ is not uniquely defined, since there is no longer a one-to-one correspondence of pushdown-store strings and states of S_g. In such a case, more than one reduction sequence may exist for a string in the acceptor's language. When these two degenerate conditions arise during the extension of a BCA, it is instructive to rewrite that BCA as a rightmost parsing algorithm to see whether the same degenerate conditions arise when parsing strings of that language from right to left.

We can next define the language accepted by a BCA in terms of a computation involving sequences of multiple configurations. We say that $C_1 \overset{*}{\vdash} C_m$ when there exists a sequence of (possibly multiple) configurations $(C_1, \ldots, C_m)$ such that

$$C_{k-1} \vdash C_k \quad \text{for } k = 2, \ldots, m \; .$$

The language of an acceptor A which is extended to handle multiple configurations is then given by

$$L(A) = \{x: (x \in T^* - \{e\}) \; \& \; [[(\# S_0 x \#) \overset{*}{\vdash} (\# F \#)$$
$$\vee \; [[(\# S_0 x \#) \overset{*}{\vdash} ((V_1, \ldots, V_n) \, S_F \#)]$$
$$\& \; (S_F \in P(Q) - Q) \; \& \; (\exists q_i) \, [(q_i \in S_F)$$
$$\& \; (q_i = F) \; \& \; (V_i = \#)]]\}$$

With these preliminaries in mind, we can state the following theorems that are proved in (13).

Theorem 1 Let P be a BCA for which $\vdash$ is uniquely defined for multiple configurations. Let

$$P' = (Q', T, N', M', \#, S_0, F)$$

with $Q' \subseteq P(Q)$, $N' \subseteq N \cup Nx \ldots xN$, and M' the original M of P together with the transitions defined on multiple configurations.

Then, $$L(P) = L(P') \; .$$

That this is so follows from the observation that, for $\vdash$ uniquely defined on multiple configurations of P, all computations of P over some input string x in L(P) are contained in a single computation of P′ over x. Conversely, no computation of P′ over some input string y will succeed unless y is in L(P).

Theorem 2 Let P be a BCA constructed from grammar G having multiple configurations for which $\vdash$ is uniquely defined. Then L(G) is unambiguous.
That this is so arises from the fact that the conditions for uniqueness of $\vdash$ also insure uniqueness of leftmost parses in L(G).

Upper bounds on storage and computation times

This concluding section contains a fairly elementary proof of the fact that BCA's can accept their languages in time directly proportional to the length of their input strings (or programs to be translated). Our reason for including this proof is to emphasize the need for a basis of comparison between different compiler-writing systems. Thus, if compiler-writing system A can produce a single-scan FORTRAN compiler whose translation speed is bounded by n^2 (with n the length of an input program), and compiler-writing system B claims a speed of, say, n log(n), it would seem fairly obvious that system C, with speed 3n, would be the economic choice for a fast, single-scan compiler system. Again, if there are limitations on computer memory size available for the compiler, and if the compiler is to run in a "load and go" or, possibly, a "reentrant" mode, it is desirable to pick a compiler-writing system that uses as little "core" as possible.

The actual *size* of one of our compilers is determined by the number of rules in the grammar of its language and, also, by the length of these rules. Roughly speaking, the number of "states" of a compiler is less than or equal to the number of rules in its normal-form grammar of the types

$$A_i \rightarrow A_{i1}A_{i2} \quad \text{and} \quad A_j \rightarrow A_{j1}a_{j2} \, .$$

The number of comparisons necessary to specify a transition away from one state is bounded by the number of contexts that can determine a transition from that state. Thus, the size of the compiler program is related to the number of rules used in the language and to their lengths.

We can speak more quantitatively about the amount of space required for the pushdown store of a deterministic BCA. Let $x = a_1 \ldots a_n$ be an input string to some BCA, and let $y = \not\# K_1 \ldots K_m$ represent the string of symbols on the pushdown store at some point during a computation. Then, after symbol $a_k (k = 1, \ldots, n)$, is "erased" by the BCA,

$$m \leq k + 1 \, .$$

This is so because, by our definition of a BCA constructed from a grammar G:

(a) For each input symbol erased, the stack can increase in length by at most one symbol.
(b) For each stack symbol erased, at most one symbol can take its place.

Hence, there are never more than (n + 1) symbols on the stack, where n is the length of the input string. In the case of an extended BCA with multiple-state configurations, there can never be more than k stacks active at once, where k is the number of states in the original BCA. For such a BCA, then, there is an upper bound of k(n + 1) symbols stored on stacks during a computation.

In order to derive an upper bound for computation time of a BCA, we must first discover an upper bound on the number of actions that can be taken by a BCA without erasing an input-string symbol during a computation. Let p be the number of rules in the grammar for a BCA such that the rules form a chain

$$A^{(i-1)} \rightarrow A^{(i)} \qquad i = 2, \ldots, p \, ,$$

such that all rules of the chain have at least one context in common, and such that p denotes the length of the longest chain of rules of this sort in the grammar. Then, without the erasure of a symbol from the pushdown store and the input string, at most p state transitions can occur.

If g_k symbols are on the pushdown store after input string symbol a_k has been erased, at most

$$(1 + g_k)(1 + p)$$

state transitions can occur when erasure of stack symbols is allowed before symbol a_{k+1} is erased. However, if only w_k symbols are removed from the stack, then at most

$$(1 + w_k)(1 + p) \quad , \quad w_k = 0, 1, \ldots, p$$

state transitions can take place before a_{k+1} is erased.

Next, we can ask what total number of symbols can be erased from the stack during any computation, i.e., what is the maximum value of

$$\sum_{k=1}^{n} w_k \quad ?$$

To answer this question, we note again that our BCA model only allows a new stack symbol to be written as a result of the erasure of an input-string symbol. Since, for an input string of length *n*, no more than *n* symbols can be written on the stack, no more than *n* symbols can be extracted from the stack during any computation. Thus,

$$\sum_{k=1}^{n} w_k \leq n \, .$$

Finally, we can arrive at an upper bound for the number of configurations that can appear during the computation of a deterministic BCA over a string x of length n:

$$\text{MAX} \leq (n + 1) + (p + 1)(w_1 + \ldots + w_n + n)$$
$$\text{or} \quad \text{MAX} \leq n(2p + 3) + 1$$

We know also that

$$\text{MAX} \geq n + 1 \, ,$$

where this lower bound is reached when the BCA of some grammar has an empty pushdown-store vocabulary; i.e., is a finite-state acceptor. Hence,

$$n + 1 \leq MAX \leq n(2p + 3) + 1$$

The upper bound on the number of configurations during a computation also holds for extended BCA's having multiple-state configurations. This is because the computations of the nondeterministic BCA from which the extended BCA was constructed are all included in the computations of that extended acceptor.

ACKNOWLEDGMENTS

The author expresses his sincere gratitude to Dr. Michael Harrison for his thoughtful criticisms and suggestions on the subject of this paper. The research for this paper was supported in part by N.A.S.A. grant NsG-398 to the Computer Science Center of the University of Maryland.

REFERENCES

1 J EICKEL M PAUL F L BAUER K SAMELSON
A syntax-controlled generator of formal language processors
Comm ACM 6 August 1963 451–455

2 R W FLOYD
Syntactic analysis and operator precedence
J ACM 10 July 1963 316–333

3 S GINSBERG S GREIBACH M HARRISON
Stack automata and compiling
J ACM 14 January 1967 172–201

4 S GINSBURG M HARRISON
One-way nondeterministic real-time list-storage languages
J ACM 15 July 1968 428–446

5 S GINSBURG
The Mathematical Theory of Context-Free Languages 1966

6 S GREIBACH
Inverses of phrase structure generators
Doctoral Dissertation Harvard University 1963

7 L H HAINES
Generation and recognition of formal languages
Doctoral Dissertation MIT 1965

8 E T IRONS
A syntax-directed compiler for ALGOL 60
Comm ACM 4 January 1961 51–55

9 D E KNUTH
On the translation of languages from left to right
Information and Control 8 December 1965 607–639

10 P M LEWIS II R E STEARNS
Syntax-directed transduction
J ACM 15 July 1968 465–488

11 C PAIR
Arbres, piles et compilation
Rev. Francaise Trait Inf 7 1964 199–216

12 A ROSENBERG
On the independence of real-time definability and certain structural properties of context-free languages
J ACM 15 October 1968 672–679

13 V B SCHNEIDER
The design of processors for context-free languages
National Science Foundation Memorandum Northwestern University August 1965

14 V B SCHNEIDER
Pushdown-store processors of context-free languages
Doctoral Dissertation Northwestern University 1966

15 V B SCHNEIDER
Syntax checking and parsing of context-free languages by pushdown-store automata
Proc S J C C 1967

16 V B SCHNEIDER
A fast translator system for the EULER programming language
Computer Science Center Technical Report University of Maryland 1969

17 N WIRTH H WEBER
EULER: A generalization of ALGOL and its formal definition: parts I and II
C ACM 3 January-February 1966 13–25 and 89–99

Generating parsers for BNF grammars*

by FRANKLIN L. DEREMER

Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

The procedure described herein is in essence an extension, albeit a simplification, of the work Earley[1] which in turn was based on Evans,[2] Feldman,[3] Floyd,[4,5] and Standish.[6] For a large subset of grammars, the procedure maps the Backus Naur Form (BNF) definition of the grammar of a language into a deterministic, left-to-right parser for the sentences in that language. It is shown below that the procedure by design, covers *all bounded right context* grammars and, as a by-product, *some* $LR(k)$ grammars which are not bounded right context. See Knuth[7] for the definitions of these classes of grammars. If two parameters are incorporated a priori into the procedure, one limiting the look-back and the other limiting the look-ahead capabilities of the parser to be generated, an algorithm results. For each BNF grammar G the algorithm either rejects G as not bounded right context for the specified limits or it generates a parser for G.

More precisely, the algorithm maps a set of BNF productions into a *program*: a "reductions analysis" program† consisting of modified Floyd productions, really *reductions*, referred to below as FPL (Floyd Production Language) statments.** The program consists of labeled, mutually exclusive groups of statements called sections. Each section has a specific task to perform. It is activated, by transfer of control to its first statement via the label, only at appropriate times. Upon each activation it either scans a new terminal symbol or makes a reduction (combined with an "unscan" in the case of a production with an empty right part) and then transfers control to the appropriate next section, or it transfers control to an ERROR routine if control falls out the bottom of the section.

The algorithm is based on Earley's intuitive notion that the top symbols on the stack matched against the right parts of certain productions should determine parsing decisions. It is an extension of his algorithm in that it provides for both finite look-ahead and finite look-back and in that it covers productions with empty right parts. It is a simplification of his algorithm in that it allows reductions only at the top of the stack, therefore reducing the number of mapping rules.

A word about notation is in order before proceeding. In this paper, non-terminal symbols are represented by Latin capitals, terminals by lower case Latin letters, arbitrary strings by the Greek letter α, and the empty string by the Greek ϵ. The Greek letter σ designates a symbol which matches any other symbol.

The algorithm

The algorithm simply consists of: (a) three rules to determine what sections are necessary for the program, (b) three rules to determine which productions should be mapped into statements for each section, (c) four rules to map the productions into statements, (d) a rule which prescribes the combination of some statements in a given section and a corresponding combination of certain sections, and (e) a contextual analysis rule for expanding statements so no two statements in a given section are both applicable to a given stack and input string configuration. The latter operation is referred to below as making the statements

* This work was supported in part by the Department of Computer Science, University of Illinois, Urbana, Illinois, and in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center, under Contract No. US AF 30(602)4144.

† It is assumed that the reader is familiar with reductions analysis programs and the associated stack, input string, and manipulations thereupon. See Cheatham.[8]

** This nomenclature is adapted to clarify the distinction between the BNF *productions*, which together define the grammar of a language, and the FPL *statements* which, when combined to form a program, describe a parser.

disjoint. There are also several rules for optimizations, some of which are given toward the end of the paper.

a. *Necessary Sections.* A special START section and a special section for SUCCESS EXIT are required together with the following:
    1. A section labeled Nh is required for each non-terminal N which appears in the right part of some production as other than the first symbol. This section is activated whenever one of the terminals, which may begin N, is supposed to be at the top of the stack. It is the purpose of section Nh to verify that one of these terminal "*h*ead symbols" is indeed at the top and, depending upon which terminal is there, to take appropriate action to commence, and perphaps conclude, a reduction to N.
    2. A section labeled t (π, p) is required for each occurrence of a *t*erminal as the p-th symbol in the right part of each production π, where where $p \geq 2$. This section consists of exactly one statement which compares the first p symbols of production π with the top p symbols of the stack. It is activated only when the match must occur for a well-formed string. Its purpose is to verify the top symbol and to take appropriate action to continue the parse.
    3. A section labeled Nt is required for each non-terminal N which appears in the right part of some production. The section is activated immediately after a reduction to N occurs at the *t*op of the stack. The statements in this section indicate comparisons to the stack to determine which of the productions in whose right part N appears is applicable to the case at hand. A match determines the appropriate subsequent action.

b. *Descriptor Sets.* In order to generate the appropriate set of statements for a given section, a descriptor set of pairs $D = ((\pi_1, p_1), \ldots)$ is associated with each section label. This descriptor set serves to indicate to which part of which productions the mapping rules described below are to be applied. The pair (π, p) points to the first p symbols of production π as the stack comparison symbols of the corresponding statement. The descriptor sets are determined as follows:
    1. D_{Nh}: Initially D_{Nh} is empty and the following recursive procedure is applied. The right part of each production π that defines N is examined. If it is empty, then $(\pi, 0)$ is added to D_{Nh}; if it begins with a terminal, then $(\pi, 1)$ is added to D_{Nh}; otherwise it begins with a non-terminal and the procedure is applied to that non-terminal.
    2. $D_{t(\pi,\ p)}$ contains exactly one pair (π, p).
    3. D_{Nt}: The right part of each production π is examined. If the non-terminal N appears as the p-th symbol, then (π, p) is in D_{Nt}.

c. *The BNF to FPL Mapping Rules.* Presented in Table I are four rules for mapping BNF productions into FPL statements. Together with the descriptor sets they represent a naive first try at generating a parser for the grammar. Implicitly, the rules assume there is no question about which production applies to the case at hand but only what action is to be taken by the parser next, given that a certain production is applicable. It is the purpose of the last two rules of the algorithm to extend it to cover a reasonable set of grammars by resolving confusion about which productions may apply to different cases within a given section.

Table I

The BNF to FPL mapping rules. α represents the first p symbols of production π, σ is a symbol which matches any other symbol, and $q = p + 1$.

BNF Production (π, p)	Maps Into	FPL Statement			
(1) M ::= α N ...	=>	$\alpha\vert$	*		Nh
(2) M ::= α b ...	=>	$\alpha\vert$	*		$t(\pi, q)$
(3) M ::= α	=>	$\alpha\vert$	→	M$\vert$	Mt
(4) M ::= ε	=>	$\sigma\vert$	→	M$\vert\sigma$	Mt

The rules of Table I are explained intuitively as follows. If the first p symbols of the right part of production π are at the top of the stack and

1. if the $(p + 1)$st symbol is a non-terminal N, then the parser should scan (*) the next terminal and activate section Nh to begin to reduce a substring to N.
2. if the $(p + 1)$st symbol is a terminal b, then the parser should scan the next terminal and activate section $t(\pi, q)$, where $q = p + 1$, to verify that that terminal is indeed

b and to decide how to continue the parse.

3. if the p-th symbol is last in the right part of the production, then the parser should make a reduction (→) to the symbol M of the left part of the production and activate section Mt to decide how to continue the parse.
4. if p = 0 and therefore the right part of the production is empty, then the parser should "unscan" the top symbol, push an M onto the top of the stack, and activate section Mt to decide how to continue the parse. The symbol unscanned will be a terminal since this statement will appear only in an Nh-type section, the activation of which is always immediately preceded by a scan (see rule (1)).

d. *Combinations.* In general, a reductions analysis program generated according to the above rules will contain sections in which some of the statements are not disjoint. That is, the conflicting statements will indicate stack comparisons (1) which are identical, or (2) the shorter of which are identical to the top few symbols of the longer ones. Thus, several statements may be applicable to a single stack and input string configuration, and the parser is in some sense non-deterministic. To render the parser deterministic it must be modified so it can either delay or determine the decisions concerning which of the several similar productions associated with the conflicting statements is applicable in various cases. Decision delays are effected by pairwise statement combinations as follows.

   If two statements in a given section are not disjoint and if each was generated according to either mapping rule (1) or (2), then they are replaced with a single statement: one whose stack comparison is the shorter of the two and which, upon a successful stack match, scans a new terminal and activates a new combination section which must be added to the program. The new section is that section whose descriptor set is the union of the two descriptor sets of the sections which the original, statements would have activated. Of course the statements in the new section must be checked for disjointness. The old sections, of which the new one is a combination, should be checked for usefulness, since the only reference in the entire program to one or both might have been deleted by removal of the two statements.

e. *Expansion by Contextual Analysis.* The only decisions which cannot be delayed are those concerning reductions. This limitation is due to the requirement that reductions be made only at the top of the stack. Thus, conflicts with statements generated according to mapping rules (3) and (4) cannot be resolved by combination. In this case the statements' comparison fields are expanded by contextual analysis to provide the parser with whatever finite look-ahead and look-back are necessary to make the decision at hand; i.e.,

   for each of the conflicting statements the grammar is investigated and generation is begun of the strings of symbols which, in the context of the production associated with the statement, may surround the original stack comparison substring α of the statement. Appropriate comparison of the composite strings associated with each of the original statements, indicates the minimum context which must be checked to make the statements disjoint. In the worst case each statement must be replaced with several statements which differ from the original in that they indicate more symbols which must be matched in the stack and/or the input string.

Examples

Since the parser proceeds from left to right, always making reductions at the top of the stack on the basis of whatever finite look-ahead and look-back are necessary, the procedure by definition covers *all bounded right context grammars.* Further, due to the fact the sections of the program themselves imply certain extra information about the stack configuration, in the same sense that a state of a finite state acceptor implies information about the string read, the algorithm also covers *some LR(k) grammars* which are not bounded right context. An example grammar in this class is S :: = aA | bB, A :: = cA | d, B :: = cB | d, the sentences of which are a c^n d and b c^n d. This grammar is not bounded right context since the clue as to whether to reduce d to A or B is an a or b arbitrarily far down the stack. The grammar is however, LR(0) and can be parsed by the algorithmically generated parser of Figure 1. In both Figures 1 and 2 a transfer of control to an ERROR routine is implicit at the bottom of each section in case no match occurs.

As an example of a grammar requiring both look-ahead and look-back consider the following LR (2) grammar.

START(Sh)	a|	*		Ah
	b|	*		Bh
Ah	c|	*		Ah
	d|	→	A|	At
Bh	c|	*		Bh
	d|	→	B|	Bt
At	cA|	→	A|	At
	aA|	→	S|	St
Bt	cB|	→	B|	Bt
	bB|	→	S|	St
St				SUCCESS EXIT

Figure 1—Algorithmically generated parser for a grammar which is LR(0) but not bounded right context

π	p 123
1	S :: = cAB
2	S :: = dA
3	A :: = aG
4	B :: = xe
5	G :: = Gx
6	G :: = x

Confusion arises in the Gt section about when to terminate the gathering of x's into the non-terminal G. Generation of the context related to production five produces three possible strings for two-symbol look-ahead:

(1) G | → G | x → G | xx

(2) G | → G | x → aG | x → daG | xe

(3) G | → G | x → aG | x → caG | xB
→ caG | xxe

There are two possible strings for production three:

(1) aG | → daG | e

(2) aG | → caG | B → caG | xe

The confusion is between case (2) of production five and case (2) of production three. One possible solution is to construct the following Gt section:

Gt	G | xx	*		t(5, 2)
	daG | xe	*		t(5, 2)
	aG |	→	A |	At

Note that advantage has been taken of the sequential nature of the program here. Since the first two statements will catch all configurations to which production five is applicable, the statement associated with production three checks no extra context. That is, the restriction that the statements in a given section must be disjoint may be relaxed in special cases where advantage is taken of the order in which statements are executed, however the contextual analysis must still be performed to ascertain the validity of such an optimization. Finally, note that had production five been G :: = xG the grammar would not have been bounded right context nor covered by this algorithm, although it would still be LR(2).

As a final, larger, and more practical example consider the grammar of Table II, which is Earley's example of a simple algebraic language. The corresponding list of necessary sections and their descriptor sets are presented in Table III, and the parser is given in Figure 2. This grammar requires no special look-back and look-ahead of more than one symbol in only one case, section Dt. A single pair of statements was combined in section Ht causing the combination of sections t (1, 2) and t (4, 2) to form a section labeled t (1, 2; 4, 2). Note that such combinations are probably most efficiently effected by operations on the descriptor sets before the sections are generated. Also note that maximum advantage was taken of ordering the statements.

For expositional purposes several optimizations were not made: (1) since the first p-1 symbols are matched immediately prior to its activation, a t(π, p) section need match only the p-th symbol with the top symbol of the stack, (2) since a reduction to N occurs immediately prior to the activation of an Nt section, it need not match the top symbol, (3) many sections could have been "concatenated" to reduce transfers and multiple interrogations, as for example sections Dt t (6, 2), and t (6, 3) which would form

Dt	D | ;r	***		Lh
	bD |	→	H |	Ht

(4) since sections Ph, Fh, and Th are identical, and are a subset of section Eh, all these could have been com-

```
START (Σ h)     ⊢|       *          B h
B h             b|r      *          D h
                b|       →    H|    H t
D h             r|       *          L h
L h             i|       →    L|    L t
T h             i|       →    P|    P t
                (|       *          E h
E h             ±|       *          T h
                i|       →    P|    P t
                (|       *          E h
S h             i|       *          t (9,2)
F h             i|       →    P|    P t
                (|       *          E h
P h             i|       →    P|    P t
                (|       *          E h
t (1,2; 4,2)    He|      →    B|    B t
                H;|      *          S h
t (6,2)         D;|      *          t (6,3)
t (8,2)         L,|      *          t (8,3)
t (9,2)         i←|      *          E h
t (12,2)        E±|      *          T h
t (14,2)        T*|      *          F h
t (16,2)        F↑|      *          P h
t (0,3)         ⊢B⊣|     →    Σ|    Σ t
t (6,3)         D;r|     *          L h
t (8,3)         L,i|     →    L|    L t
t (18,3)        (E)|     →    P|    P t
H t             H|       *          t (1,2; 4,2)
D t             D|;r     *          t (6,2)
                bD|      →    H|    H t
L t             L|,      *          t (8,2)
                D;rL|    →    D|    D t
                rL|      →    D|    D t

T t             T|*      *          t (14,2)
                E±T|     →    E|    E t
                ±T|      →    E|    E t
                T|       →    E|    E t
E t             E|±      *          t (12,2)
                (E|      *          t (18,3)
                i←E|     →    S|    S t
F t             F|↑      *          t (16,2)
                T*F|     →    T|    T t
                F|       →    T|    T t
P t             F↑P|     →    F|    F t
                P|       →    F|    F t
B t             ⊢B|      *          t (0,3)
S t             H;S|     →    H|    H t
Σ t                                 SUCCESS EXIT
```

Figure 2—Algorithmically generated parser for a simple algebraic language

bined to save space; however this is probably undesirable as it implies a loss of information useful for error recovery, and (5) often the knowledge of one symbol on the stack implies the knowledge of several below it: for instance, an i is always below ←, and F is always below an ↑, etc.

If the first, second, and fifth of these optimizations are applied to the program of Figure 2, the result is a program with an average of less than one symbol match indicated per statement. Since there is an average of 1.7 statements per section and since each section either scans a new symbol or makes a reduction, it can be concluded that the parsing time required for any string of length n, requiring p reductions to parse it, is at most roughly:

1.7 * (n + p) * (the time to compare two symbols)

+ n * (the time to scan a symbol)

+ p * (the time to make a reduction)

+ (n + p) * (the time to transfer to a new section)

Table II

Production Table for a Simple Algebraic Language

	π			p 1	2	3	4
<AXIOM>	0	Σ	::=	⊢	B	⊣	
<BLOCK>	1	B	::=	H	e		
<HEAD>	2	H	::=	b			
	3	H	::=	b	D		
	4	H	::=	H	;	S	
<DECLARATION>	5	D	::=	r	L		
	6	D	::=	D	;	r	L
<TYPE LIST>	7	L	::=	i			
	8	L	::=	L	,	i	
<STATEMENT>	9	S	::=	i	←	E	
<EXPRESSION>	10	E	::=	T			
	11	E	::=	±	T		
	12	E	::=	E	±	T	
<TERM>	13	T	::=	F			
	14	T	::=	T	*	F	
<FACTOR>	15	F	::=	P			
	16	F	::=	F	↑	P	
<PRIMARY>	17	P	::=	i			
	18	P	::=	(	E	)	

NOTE: i is identifier

r is <u>real</u>

b is <u>begin</u>

e is <u>end</u>

If the third optimization is applied, the first and last of the terms in this sum can be reduced. Of course, the coefficient of the first term cannot be less than one and the second and third terms cannot be reduced. Clearly, the algorithm with optimizations generates a practical parser for this special case.

Table III

List of Necessary Sections and the Corresponding Descriptor Sets for the Grammar of Table II

NECESSARY SECTIONS	DESCRIPTOR SETS			
S T A R T (Σ h)	0,1			
B h	2,1	3,1		
D h	5,1			
L h	7,1			
T h	17,1	18,1		
E h	11,1	17,1	18,1	
S h	9,1			
F h	17,1	18,1		
P h	17,1	18,1		
t (1,2)	} combined to form t (1,2; 4,2)			
t (4,2)				
t (6,2)				
t (8,2)				
t (9,2)				
t (12,2)				
t (14,2)				
t (16,2)				
t (0,3)				
t (6,3)				
t (8,3)				
t (18,3)				
H t	1,1	4,1	(combination)	
D t	6,1	3,2		
L t	8,1	5,2	6,4	
T t	10,1	14,1	11,2	12,3
E t	12,1	18,2	9,3	
F t	13,1	16,1	14,3	
P t	15,1	16,3		
B t	0,2			
S t	4,3			
Σ t	----			

CONCLUSION

Hand simulation of the algorithm for grammars of up to eighty-five productions indicates that results comparable to the above can be expected for typical programming languages. For instance, grammars which are "mostly" simple or operator-precedence produce parsers which operate in times comparable to the above and whose sizes are proportional to the size of the grammar (about two to four times as many FPL statements as BNF productions).

A modified version of the algorithm has been implemented as part of a translator writing system for the ILLIAC IV computer at the University of Illinois. The system and the modifications to the algorithm are described in detail in Beals.[9] A preliminary result is a *recognizer* generated for an ALGOL-like language about one quarter the size of ALGOL which processes 850 cards per minute. This device goes through the motions of lexical analysis and parsing, but it drives only enough semantic routines to check scopes of variables, types, etc.

Although a proof has not been given, it seems intuitively reasonable that the algorithm generates correct parsers. The author believes that the methods of Evans could, without extreme difficulty, be utilized to provide such a proof.

ACKNOWLEDGMENT

The author would like to thank Alan J. Beals for his help in evaluating and debugging the algorithm. Thanks are also due Dr. R. S. Northcote for suggested improvements in the paper itself.

REFERENCES

1 J EARLEY
Generating a recognizer for a BNF agrmmar
Carnegie-Mellon Institute of Technology 1965 upnublished

2 A EVANS
Syntax analysis by production language
Doctoral Thesis Carnegie-Mellon Institute of Technology 1965

3 J FELDMAN
A formal semantics for computer-oriented languages
Doctoral Thesis Carnegie Mellon Institute of Technology 1964

4 R FLOYD
A descriptive language for symbol manipulation
J A C M Vol 8 No 4 579–584 1961

5 R FLOYD
Bounded context syntactic analysis
C A C M Vol 7 No 2 62–67 1964

6 T STANDISH
Generating productions from a restricted class of BNF grammars
Carnegie-Mellon Institute of Technology Computation Center unpublished

7 D KNUTH
On the translation of languages from left to right
Information and Control Vol 8 607–639 1965

8 T CHEATHAM
The theory and construction of compilers
Massachusetts Computers Associates Inc 1967

9 A BEALS
The generation of a deterministic parsing algorithm
ILLIAC IV Document No 304 University of Illinois 1969

An extended BNF for specifying the syntax of declarations

by GORDON WHITNEY

Western Electric Engineering Research Center
Princeton, New Jersey

"I recommend the use of suitable special metalanguages as a part of the defining reports. The Backus notation is probably as good as anything we have at present, but it still leaves a great deal to be desired . . . many syntactic rules cannot be expressed in it. The language reporter should . . . invent and use such tools wherever the special metalanguage is easier to work with than natural language." Peter Naur,[1] 1963.

INTRODUCTION

The syntax of ALGOL is specified in its defining report[2] by a grammar[3] whose rules are given in Backus-Naur Form (BNF). Because of the need for precision in the specification of complex systems, BNF has been used to record the syntax of many other programming languages. The rules of BNF are equivalent to context-free[4] production rules. However, due to its declarations, ALGOL is context-dependent, and its syntax cannot be fully specified by a grammar limited to context-free rules.[5] Other grammars for ALGOL[6,7,8] are more compact than the BNF of the defining report, but they are generatively no more powerful. Because of this context-dependency of declarations, even an assembly language cannot be fully specified in BNF.

This paper defines an extension to BNF which permits the specification of the syntax of declarations, while retaining the definitional power of BNF as a subset. PL/I has been formally defined[9] by specifying a model which will execute programs written in the language. This model specifies the semantics of the language but does not include a grammar for the syntax of declarations. Recent papers[10,11,12,13] define systems capable of specifying the syntax of declarations. This paper is an extension and exposition of one of these models* (called "table-grammar") in the hope that language reporters and implementors will be able to utilize the model in practical work with programming languages. Consistent with this purpose, a system of notation has been adopted (with an attendant loss of compactness) which is both compatible with BNF and oriented for the human reader of the syntax.

Table-language generation

A grammar defines a language by specifying the means by which legal strings in the language can be generated starting from some fixed initial configuration. For each class of language there exists a corresponding class of processors which can carry out the generative process. Such a processor is called a generator. This section will define the form of a table-grammar and the means by which they generate strings.

* The table-grammar presented here is similar but not identical to a previous model.[13] The differences are as follows:

1. The table-functions using Greek letters as function names have been changed to expression format using operators with English names. Thus the τ-function is replaced by the use of *create, retrieve* or *recreates*. The ρ-function becomes *replaces*. The μ-function becomes *illegal*.
2. The use of two auxiliary storage tapes rather than one gives the new model increased generative power. This also allows BNF grammars to remain nearly intact when table operations are added to the grammar. The table operations *copytable* and *erasetable* are new and are necessary to allow multiple tables in a two-tape system.
3. The definition of the generation operators to perform recognition is new, as is the operator *copy* and both the definitions and the properties of "Regular Table Expressions."

Notation

Certain definitions will be used throughout. They will not be given further local definition. These definitions are:

a, b, c, . . .	terminals symbols.
ϵ	a terminal-string of length zero.
x, y, z	terminal-strings, having an arbitrary value, or a range of values.
+	this post-fix operator denotes ϵ-free closure for catenation, i.e., $(X)^+ = \{ \bigcup_{i=1}^{\infty} X^i$ where $X^1 = X$ and $X^{i+1} = X^i X\}$.
*	this post-fix operator denotes full closure for catenation, i.e., $(X)^* = \{(X)^+ \cup \epsilon\}$.
λ	this symbol denotes a string of either letters or blanks and is called a letter-string.

Definition of a table-grammar

A table-grammar is a 4-tuple:

G = (categories, terminals, static-rules, sentence-symbol)

where:

1. The set of categories and the set of terminals are finite and disjoint.
2. The set of categories has two disjoint subsets: grammar-categories and table-categories.
3. The set of static-rules is finite. Each rule has the form: $\langle\lambda\rangle$ :: = (grammar-category $\cup$ terminal $\cup$ table-expression)* where:
   a. The *leftside* of a rule is a grammar-category.
   b. The *rightside* is called an *arbitrary-string*.
4. A table-expression consists of a generation-operator with its respective operands. A generation-operator has either zero, one or two operands, and is respectively referred to as a niladic, monadic or dyadic operator. The operands for table-expressions are table-categories whose form is $\langle\cdot\lambda\cdot\rangle$. For the presentation of table-expression schema it will be convenient to use $\langle\cdot i\cdot\rangle$ to represent an incident table-category (i.e., one entering the table) and $\langle\cdot r\cdot\rangle$ to represent a resident table-category (i.e., one already present in the table). The following are the schemas for the seven different forms of table-expressions:
   a. The niladic expressions are: *copytable* and *erasetable*.
   b. The monadic expressions are: *create* $\langle\cdot i\cdot\rangle$ *retrieve* $\langle\cdot r\cdot\rangle$ and *illegal* $\langle\cdot r\cdot\rangle$.
   c. The dyadic expressions are:

      $\langle\cdot i\cdot\rangle$*recreates* $\langle\cdot r\cdot\rangle$ and $\langle\cdot i\cdot\rangle$*replaces* $\langle\cdot r\cdot\rangle$.

   Table expressions, when evaluated, have the following characteristics:
   d. Successful evaluation may be subject to certain restrictions.
   e. A terminal-string is returned as the value of the expression.
   f. Side-effects may be produced in the active-table. (The concept of active-table will be defined below.)
5. The sentence-symbol is a unique grammar-category.

Concerning the evaluation of table-expressions, if the specified conditions are not satisfied, the current generation step is blocked and an alternate sequence of steps must be utilized. Thus invalid programs are eliminated at the point in the generation sequence where an incorrect construction would have appeared.

String processing devices

A string processing *device* is a machine with a finite control, with a system of one or more auxiliary tapes providing potentially infinite storage, and with either an input-tape or with an output-tape or with both. The device is called *deterministic* if in every possible configuration it has only one possible move to its next configuration. If the device has more than one possible move, it is called *nondeterministic*. A device with an input-tape is called a recognizer. A device with an input and output tape is called a transducer. A device with only an output-tape is called a generator. The sets defined by a context-free grammar are exactly the sets generated by some nondeterministic generator whose auxiliary storage is a pushdown and whose output-tape is one way.

If an input (or output) tape of a device is one-way (usually this is left to right) then the device is called on-line and its input (or output) need not be written on a tape at all but need only be received (or transmitted) one character at a time without the use of any local storage.

A table-language generator

Figure 1 shows the type of device necessary to generate table-languages. The output is one way and the output-string is not counted as a tape. In a one pass processor, as declarations of identifiers are encountered, they must be recorded in some form of auxiliary storage to allow for subsequent retrieval. The table-tape stores a sequence of declaration tables as a pushdown list of

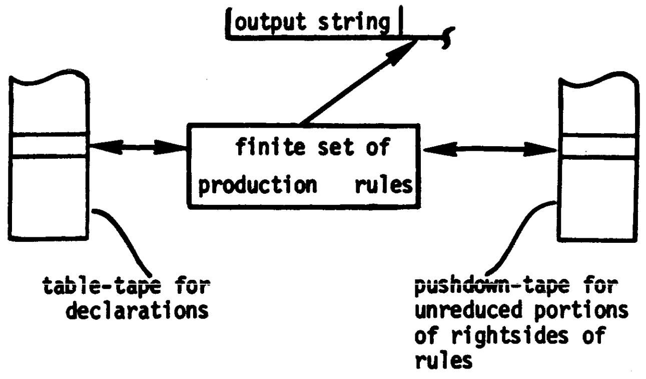

output string
finite set of production rules
table-tape for declarations
pushdown-tape for unreduced portions of rightsides of rules

Figure 1—A nondeterministic two-tape generator for a language defined by a table-grammar

tables. The top most table is designated as the active-table. When a block structure is not required, only the active-table will appear on the table-tape. The operation of the table is subject to four restrictions:

1. Entries must be dynamic-rules of the form:

   $(\langle\cdot\lambda\cdot\rangle \rightarrow$ identifier$)$.

2. $\langle\cdot\lambda\cdot\rangle$ is a table-category. A simple set of table-categories would be

   $\{\langle\cdot$integer$\cdot\rangle, \langle\cdot$real$\cdot\rangle\ \langle\cdot$Boolean$\cdot\rangle\}$.

3. An identifier is an element from a regular set. This means that an identifier can be recognized by a finite automaton. A simple set would be ⟨letter⟩ (⟨digit⟩)* where a3 and j49 would be identifiers.
4. The identifiers within the active-table must be unique. The same identifier may not appear twice. Note by way of contrast that the same table-category may appear many times. Thus:

   $(\langle\cdot$real$\cdot\rangle \rightarrow$ a56$)(\langle\cdot$real$\cdot\rangle \rightarrow$ j4$)$

   is a valid table; while

   $(\langle\cdot$real$\cdot\rangle \rightarrow$ a56$)(\langle\cdot$integer$\cdot\rangle \rightarrow$ a56$)$

   is not a valid table.

An identifier is said to be *declared* if it appears in the active-table, otherwise it is *undeclared.* Each new declaration which is added to the active-table must contain an identifier which is undeclared.

Table-grammar generation sequences

The operator *copytable* is explained in the section "*Table operators for embedded blocks*". When this operator is not used in a grammar, only a single table copy will appear on the table-tape. The description of generation sequences for this portion of the paper will be limited to those generations in which only one table copy is present on the table-tape. A generation sequence is a finite list of transitive steps. Each step has the form:

$$ID_1 \Rightarrow ID_2$$

where ID stands for instantaneous description and where $\Rightarrow$ indicates the rewriting of ID_1 in accordance with an evaluation algorithm to produce (i.e., generate) ID_2. An instantaneous description is a triple of the form:

(terminal-string, arbitrary-string,
table-configuration)

where arbitrary-string represents the catenation of unreduced rightsides of static-rules and where a table-configuration is one distinct arrangement of declarations out of a potentially infinite number of such configurations designated T_0, T_1, T_2, ... In actual sequences, the parentheses and commas may be omitted with no loss of meaning by the use of disjoint alphabets. The active-table is erased on the last step of a generation sequence.

The evaluation algorithm for a step has three parts:

1. If $ID_1 = (x, a\gamma, T_i)$, then $ID_2 = (xa, \gamma, T_i)$.
2. If $ID_1 = (x, \langle a\rangle\gamma, T_i)$ and $\langle a\rangle ::= \beta$ is a static-rule of G, then one possible value is $ID_2 = (x, \beta\gamma, T_i)$.
3. If $ID_1 = (x,$ table-expression $\gamma, T_i)$ and the table-expression is such that given an active-table T_i, the expression returns as its value y and as a side-effect changes T_i to T_j, then $ID_2 = (xy, \gamma, T_j)$.

It is now possible to give a precise definition of a table-language.

Definition of a table-language

A table-language is a set of strings generated by a function L having two arguments:

L(G, R) = {x where G is a table-grammar and R is a regular set, and where (ϵ, sentence-symbol, T_0) $\overset{*}{\Rightarrow}$ x, using the static-rules of G and using R as an identifier set, and where $\overset{*}{\Rightarrow}$ is the transitive closure of $\Rightarrow$ }.

Formal properties

Table-languages can be defined by means of a number of alternative models. The formal properties listed below pertain only to the model defined above with a unified identifier set and where the identifiers in the table are isomorphic to the terminal identifiers:

1. The languages are properly contained in the context-sensitive languages.
2. The languages properly contain the context-free languages.
3. The languages are closed under *, homomorphism, and intersection with a regular set.
4. The languages are not closed under union, catenation, or inverse homomorphism.

Compound rules

The static-rule schemas given above can be called *simple*. BNF uses a rule schema which will be called *compound*. If $\langle a \rangle ::= \beta$ and $\langle a \rangle ::= \gamma$ are rules in a grammar then they can be replaced by the single compound rule $\langle a \rangle ::= \beta | \gamma$. Compound rules will be used where convenient. They are generatively no more powerful than simple rules.

Table entry and access

Two basic table operations are the recording and the retrieval of a declaration. These operations are achieved by the expressions *create* $\langle \cdot i \cdot \rangle$ and *retrieve* $\langle \cdot r \cdot \rangle$ respectively, where $\langle \cdot i \cdot \rangle$ stands for an incident table-category (i.e., one entering the table) and $\langle \cdot r \cdot \rangle$ stands for a resident table-category (i.e., one already present in the table).

Table-entry

The operator *create* is employed to specify the syntax of the declaration of an identifier. The expression *create* $\langle \cdot i \cdot \rangle$, when evaluated, has the following characteristics:

1. The evaluation cannot be blocked if any additional undeclared identifiers exist.
2. The selected-identifier must be undeclared and is returned as the value which replaces the invoking expression within the intermediate-string of the generation sequence. (This will be made more explicit in the sequel by examples.)
3. The dynamic rule, $\langle \cdot i \cdot \rangle \rightarrow$ selected-identifier, is added to the top of the active-table.

In summary, the expression *create* $\langle \cdot i \cdot \rangle$ when evaluated selects an undeclared identifier, places it in the output-string and adds to the active-table the indicated declaration.

Table-access

The operator *retrieve* is used to specify the syntax of an identifier which can appear only if it has been previously declared. The expression *retrieve* $\langle \cdot r \cdot \rangle$, when evaluated, has the following characteristics:

1. The evaluation is blocked if the specified resident category $\langle \cdot r \cdot \rangle$ does not appear in the active-table.
2. The identifier in the selected dynamic rule is returned as the value.
3. There are no side-effects in the active-table.

In summary, the expression *retrieve* $\langle \cdot r \cdot \rangle$ when evaluated selects a dynamic rule whose leftside is $\langle \cdot r \cdot \rangle$ and places in the output-string the identifier appearing on the rightside of the selected rule.

Examples of context-sensitive table-languages

The context-sensitive set with strings of the form $w_1 d w_2$ where $w_1 = w_2$ and where w_1 and w_2 are in the set $(0 \cup 1)^+$ can be generated by the table-grammar:

$$\langle a \rangle ::= \textit{create}\langle \cdot b \cdot \rangle \langle c \rangle$$
$$\langle c \rangle ::= d\ \textit{retrieve}\langle \cdot b \cdot \rangle$$

where the identifier set is $(0 \cup 1)^+$. A typical sequence is:

$$\langle a \rangle T_0 \Rightarrow 01101 \ \langle c \rangle T_1 \Rightarrow 01101d01101$$

where $T_1 = ((\langle \cdot b \cdot \rangle \rightarrow 01101)$.

The context-sensitive set with strings of the form $w_1 w_2 \ldots w_n$ where $w_i \neq w_j$ for all $i \neq j$ and where w_i is in the set $(0 \cup 1)^+ d$ is generated by the table-grammar:

$$\langle a \rangle ::= \textit{create}\langle \cdot b \cdot \rangle \langle c \rangle \mid \textit{create}\langle \cdot b \cdot \rangle \langle a \rangle$$
$$\langle c \rangle ::= \epsilon$$

where the identifier set is $(0 \cup 1)^+ d$. A typical sequence is:

$$\langle a \rangle T_0 \Rightarrow 010d \ \langle a \rangle T_1 \Rightarrow 010d1011d \ \langle a \rangle T_2 \Rightarrow 010d1011d11d \ \langle c \rangle T_3 \Rightarrow 010d1011d11d$$

where $T_1 = ((\langle \cdot b \cdot \rangle \rightarrow 010d)$,
$T_2 = ((\langle \cdot b \cdot \rangle \rightarrow 1011d)((\langle \cdot b \cdot \rangle \rightarrow 010d)$, and
$T_3 = ((\langle \cdot b \cdot \rangle \rightarrow 11d)((\langle \cdot b \cdot \rangle \rightarrow 1011d)((\langle \cdot b \cdot \rangle \rightarrow 010d)$.

Table operators for contextual declarations

Programming languages utilize three different methods of associating table-categories with identifiers: explicit declaration, implicit declaration and contextual declaration. Explicit declaration of variables is used in Algol to assure that the declaration of a variable precedes its use. Syntax for explicit declaration preceding use can be specified by the operator sequence:

$$\textit{create}\langle \cdot a \cdot \rangle \ldots \textit{retrieve}\langle \cdot a \cdot \rangle .$$

An instance of contextual declaration, is the association of the category "label" with an identifier by means of the context in which the identifier appears. Two distinct contexts are used to cause such a declaration and these can be meanfully distinguished by the use of the table-categories ⟨·proper label·⟩ and ⟨·improper label·⟩. For example, if the integer "25" appears in columns 1 to 5 of a Fortran statement, then the integer "25" has been contextually declared to be a ⟨·proper label·⟩. On the other hand, if the integer "25" appears in a GO TO statement prior to its declaration as a ⟨·proper label·⟩, then the integer "25" will initially be declared as an ⟨·improper label·⟩. Two additional operators, *recreates* and *illegal*, are provided to handle the table activity associated with this kind of declaration.

Altering a table-entry

The operator *recreates* is used to alter a table-category as a side-effect of generating a declared identifier. The expression ⟨·i·⟩*recreates*⟨·r·⟩, when evaluated, has the following characteristics:

1. The evaluation is blocked if the specified resident category ⟨·r·⟩ does not appear in the active-table.
2. The identifier in the selected dynamic-rule is returned as the value.
3. In the selected dynamic-rule, the incident category ⟨·i·⟩ replaces the resident category ⟨·r·⟩.

In summary, the expression ⟨·i·⟩*recreates*⟨·r·⟩ when evaluated selects a dynamic-rule whose leftside is ⟨·r·⟩, changes ⟨·r·⟩ to ⟨·i·⟩, and places in the output-string the identifier appearing on the rightside of the selected rule.

Scan for an invalid category

The operator *illegal* is used to specify a scan for an illegal table-category. The expression *illegal*⟨·r·⟩, when evaluated, has the following characteristics:

1. The evaluation is blocked if the specified resident category ⟨·r·⟩ is present in the active-table.
2. The value returned is the empty-string, ϵ.
3. There are no side-effects in the active-table.

In summary, the expression *illegal*⟨·r·⟩ when evaluated is blocked if ⟨·r·⟩ is in the active-table, otherwise it returns ϵ and has no side-effects.

Example of a micro-assembly language (MAL)

An example in the form of a micro-assembly language (abbreviated MAL) will be used to illustrate the definitional power of the four operators given above. It will further show how a BNF grammar can be altered to become a table-grammar while preserving most of the original grammar-categories. The resulting table-grammar is only slightly larger than the original BNF grammar. MAL is not intended to be useful as a programming language. A free-form syntax has been chosen to permit a definition which is not sensitive to blanks or lines. MAL has "dc" for "define constant" and "ds" for "define storage." Statements are terminated by semicolons, labels have a colon suffix, and the period is used as a separator.

Sample program in Micro-Assembly Language (MAL)

```
a: dc . 12;
b: ds;
c: load . a;
   store . b;
   goto . c;
   end;
```

BNF grammar for MAL

The syntax of BNF can be defined by the following BNF rules:

```
⟨program⟩ ::= ⟨body⟩ end;
⟨body⟩ ::= ⟨statement⟩; | ⟨statement⟩; ⟨body⟩
⟨statement⟩ ::= ⟨declarative statement⟩ |
   ⟨imperative statement⟩
⟨declarative statement⟩ ::=
   ⟨data label⟩ : dc . ⟨integer⟩ |
   ⟨data label⟩: ds
⟨imperative statement⟩ ::=
   ⟨imperative label⟩ : ⟨unlabeled imperative⟩ |
   ⟨unlabeled imperative⟩
⟨unlabeled imperative⟩ ::=
   goto . ⟨imperative label⟩ |
   ⟨operation⟩ . ⟨data label⟩
⟨operation⟩ ::= load | add | store
⟨imperative label⟩ ::= ⟨name⟩
⟨data label⟩ ::= ⟨name⟩
⟨name⟩ ::= ⟨letter⟩ | ⟨name⟩⟨digit⟩ |
   ⟨name⟩⟨letter⟩
⟨integer⟩ ::= ⟨digit⟩ | ⟨digit⟩⟨integer⟩
⟨letter⟩ ::= a|b|z
⟨digit⟩ ::= 0|1|9
```

Semantic restraints on MAL programs

Certain restraints on MAL programs have in the past been classed as "semantics." However their formalization by a table-grammar shows them to be syntactic though not context-free. These restraints are:

1. All labels must be unique.

2. ⟨data label⟩ must be declared by a dc or ds statement before it is used in an ⟨imperative statement⟩. The op-codes "load," "add" and "store" must reference a ⟨data label⟩ not an ⟨imperative label⟩.
3. A "goto" op-code must reference an ⟨imperative label⟩, not a ⟨data label⟩ or an improper label (i.e., one for which there is no proper label declaration).

Unrestrained BNF generations

The following generations of illegal programs are possible using the BNF grammar for MAL when these so called "semantic" restraints are not observed:

Generation	*Error*
⟨program⟩ $\overset{*}{\Rightarrow}$ a:ds; a:ds; end;	Labels are not unique.
⟨program⟩ $\overset{*}{\Rightarrow}$ goto.a; end;	Label "a" is not properly declared.
⟨program⟩ $\overset{*}{\Rightarrow}$ a:ds; goto.a; end;	The goto references a ⟨data label⟩.
⟨program⟩ $\overset{*}{\Rightarrow}$ a:ds; load.b; end;	⟨data label⟩ "b" not declared.

The table-grammar for MAL

The following table-grammar for MAL is a revision of the BNF grammar given above. Rules preceded by a * have been altered. Unmarked rules are retained unchanged. Rules which do not appear have been omitted.

```
*⟨program⟩ ::=
     ⟨body⟩ illegal⟨·improper label·⟩ end;
  ⟨body⟩ ::= ⟨statement⟩ ; | ⟨statement⟩ ; ⟨body⟩
  ⟨statement⟩ ::= ⟨declarative statement⟩ |
     ⟨imperative statement⟩
*⟨declarative statement⟩ ::=
     create⟨·data label·⟩ : dc . ⟨integer⟩ |
     create⟨·data label·⟩ : ds
*⟨imperative statement⟩ ::= ⟨unlabeled imperative⟩ |
     create⟨·imperative label·⟩ : ⟨unlabeled imperative⟩|
     ⟨·imperative label·⟩recreates⟨·improper label·⟩ :
        ⟨unlabeled imperative⟩
*⟨unlabeled imperative⟩ ::= goto . create⟨·improper
        label·⟩ |
     goto . retrieve⟨·improper label·⟩ |
     goto . retrieve⟨·imperative label·⟩ |
     ⟨operation⟩ . retrieve⟨·data label·⟩
  ⟨operation⟩ ::= load | add | store
  ⟨integer⟩ ::= ⟨digit⟩ | ⟨digit⟩⟨integer⟩
  ⟨digit⟩ ::= 0|1|9
```

The categories ⟨name⟩ and ⟨letter⟩ have been deleted from the BNF grammar and are replaced by the identifier set ⟨letter⟩(⟨letter⟩ ∪ ⟨digit⟩)*. Identifiers then are any alphameric string beginning with a letter. There are three table-categories: ⟨·data label·⟩, ⟨·imperative label·⟩ and ⟨·improper label·⟩. The categories data label and imperative label have been changed from grammar-categories to table-categories and their original BNF rules have been deleted. The following list itemizes the way in which the table-grammar satisfies the three "semantic" restraints given above for MAL programs:

1. Identifiers are generated only by table-expressions. Uniqueness of declared identifiers is assured by the definition of the operator *create*.
2. The table-category ⟨·data label·⟩ is entered into the table only by the rule for ⟨declarative statement⟩. Declaration must precede use because in the rule for ⟨unlabeled imperative⟩, the grammar-category ⟨operation⟩ refers to an identifier only by accessing the table through "*retrieve*⟨·data label·⟩" and *retrieve* returns as its value only identifiers previously declared.
3. In the rule for ⟨unlabeled imperative⟩ the following cases are handled:

   a. *create*⟨·improper label·⟩ allows a label to enter the table before it is properly declared.
   b. *retrieve*⟨·improper label·⟩ allows a label which was the argument of a prior goto, to be retrieved as a reference by another goto, still prior to its proper declaration.
   c. *retrieve*⟨·imperative label·⟩ allows a label which has been previously properly declared to be used as the argument of a goto.

   In the rule for ⟨imperative statement⟩ two additional cases are handled:

   d. *create*⟨·imperative label·⟩ allows the contextual declaration of a label to precede its use in a goto.
   e. ⟨·imperative label·⟩*recreates*⟨·improper label·⟩ allows a label previously referred to by a goto to be redeclared as a proper label.

   In the rule for ⟨program⟩ one final case is handled:

   f. *illegal*⟨·improper label·⟩ blocks the generation sequence in a case when goto references have not been redeclared as a proper label.

Table operators for embedded blocks

Certain programming languages allow a block structure in which declarations outside of a block have a scope which extends inward, while declarations within a block are local to the block. An identifier already declared outside a block may be redeclared within the block to refer to a new address which will not agree with the one assigned to the same identifier when used outside the block. These requirements indicate that the table-tape must be able to operate as a pushdown store in which tables are handled as units of information. The top most table is designated the "active-table." All operators access only this table, while lower copies are inaccessible to the operators.

Table-tape pushdown operations

The niladic operators *copytable* and *erasetable* are used to control the table-tape as a pushdown store whose unit of information is a table. When evaluated, these operators have the following characteristics:

1. They cannot be blocked.
2. They return as a value the empty-string, ϵ.
3. *copytable* has the side-effect of placing a duplicate copy of the active-table on the top of the table-tape.
4. *erasetable* has the side-effect of erasing the active-table from the top of the table-tape.

Control of local and nonlocal identifiers

The need for a special operator to provide initialization for nonlocal identifiers is best explained by an example. Consider a table-grammar having only the table-category ⟨·real·⟩, and whose identifiers are alphameric strings beginning with a letter. Let a generation sequence proceed until a new inner block has just been initialized, a *copytable* has just been executed, and the table-tape contains T_1 T_1. Horizontal snapshots of the table-tape are oriented so that the active-table is on the left.

$$\begin{aligned} T_1 = {} & (\langle\cdot\text{real}\cdot\rangle \rightarrow \text{a5}) \\ & (\langle\cdot\text{real}\cdot\rangle \rightarrow \text{cd}). \end{aligned}$$

Let the active-table T_1 be updated via *create* ⟨·real·⟩ so that the table-tape contains T_2T_1 and

$$\begin{aligned} T_2 = {} & (\langle\cdot\text{real}\cdot\rangle \rightarrow \text{k9}) \\ & (\langle\cdot\text{real}\cdot\rangle \rightarrow \text{a5}) \\ & (\langle\cdot\text{real}\cdot\rangle \rightarrow \text{cd}). \end{aligned}$$

Within the active-table there is no information which indicates that k9 is local and that both a5 and cd are nonlocal to the current block. In order that such a distinction could be achieved, a new expression must be defined. This expression is ⟨·i·⟩*replaces*⟨·r·⟩, which when evaluated has the following characteristics:

1. The evaluation cannot be blocked.
2. The value returned is the empty-string, ϵ.
3. As a side-effect, each instance of the specified resident category ⟨·r·⟩, within the active-table, is changed to the specified incident-category ⟨·i·⟩.

Returning to the example given at the beginning of this section, if the expression ⟨·nonlocal real·⟩*replaces* ⟨·real·⟩ were evaluated following *copytable*, then the table-tape contains T_3T_1 and

$$\begin{aligned} T_3 = {} & (\langle\cdot\text{nonlocal real}\cdot\rangle \rightarrow \text{a5}) \\ & (\langle\cdot\text{nonlocal real}\cdot\rangle \rightarrow \text{cd}). \end{aligned}$$

A subsequent evaluation of *create*⟨·real·⟩ would give a table-tape of T_4T_1

$$\begin{aligned} T_4 = {} & (\langle\cdot\text{real}\cdot\rangle \rightarrow \text{k9}) \\ & (\langle\cdot\text{nonlocal real}\cdot\rangle \rightarrow \text{a5}) \\ & (\langle\cdot\text{nonlocal real}\cdot\rangle \rightarrow \text{cd}). \end{aligned}$$

In T_4, the desired distinction between local and nonlocal identifiers has been obtained.

Examples of embedded blocks

The BNF grammar

$$\langle a\rangle ::= [\langle c\rangle] \mid [\langle c\rangle\ \langle a\rangle] \qquad (1)$$

$$\langle c\rangle ::= \langle d\rangle \mid \langle d\rangle, \langle c\rangle \qquad (2)$$

$$\langle d\rangle ::= 0 \mid 1 \mid 1\langle d\rangle \mid 0\langle d\rangle \qquad (3)$$

generates the strings:

$$\langle a\rangle T^0 \overset{*}{\Rightarrow} [0, 1, 101\ [1, 010, 0]] \qquad (4)$$

$$\langle a\rangle T^0 \overset{*}{\Rightarrow} [0, 1, 101\ [101, 101]]\ . \qquad (5)$$

Now add the restraint that values for ⟨d⟩ appearing in the rule for ⟨c⟩ may appear only once at each level of self-embedding. This is equivalent to the restriction that an identifier be declared only once on each level and that identifiers declared on an outer level may be redeclared on an inner level. Given this restraint, generation (4) is well formed but generation (5) is not, because 101 is repeated on the lowest level. The following table-grammar generates (4) but will not generate (5) thus satisfying the restraint stated above.

$$\begin{aligned} \langle a\rangle ::= {} & [\textit{copytable}\ \langle b\rangle\ \textit{erasetable}] \mid \\ & [\textit{copytable}\ \langle b\rangle\ \langle a\rangle\ \textit{erasetable}] \end{aligned} \qquad (*1)$$

$$\langle b\rangle ::= \langle\cdot\text{nonlocal}\cdot\rangle \textit{replaces} \langle\cdot\text{local}\cdot\rangle\ \langle c\rangle \qquad (6)$$

$$\langle c \rangle ::= \langle d \rangle \mid \langle d \rangle, \langle c \rangle \qquad (2)$$

$$\langle d \rangle ::= \textit{create}\,\langle\cdot\text{local}\cdot\rangle \mid \langle\cdot\text{local}\cdot\rangle\textit{recreates}\,\langle\cdot\text{nonlocal}\cdot\rangle \qquad (*3)$$

Note that (*1) and (*3) are revisions for the table-grammar of (1) and (3) respectively, and that the table-category ⟨·nonlocal·⟩ enters the active-table in the rule for ⟨b⟩ and is utilized in the rule for ⟨d⟩.

The following is the table-grammar generation sequence for the string given in (4) above.

$$\langle a \rangle T_0 \overset{*}{\Rightarrow} [\,\langle b \rangle\ \langle a \rangle\ \textit{erasetable}]\ T_0\ T_0 \qquad (7)$$

$$\overset{*}{\Rightarrow} [0, 1, 101\ \langle a \rangle\ \textit{erasetable}]\ T_1\ T_0 \qquad (8)$$

$$\Rightarrow [0, 1, 101\ [\,\langle b \rangle\ \textit{erasetable}]\ \textit{erasetable}]\ T_1\ T_1\ T_0 \qquad (9)$$

$$\Rightarrow [0, 1, 101\ [\,\langle c \rangle\ \textit{erasetable}]\ \textit{erasetable}]\ T_2\ T_1\ T_0 \qquad (10)$$

$$\overset{*}{\Rightarrow} [0, 1, 101\ [1, 010, 0\ \textit{erasetable}]\ \textit{erasetable}]\ T_3\ T_1\ T_0 \qquad (11)$$

$$\overset{*}{\Rightarrow} [0, 1, 101\ [1, 010, 0]] \qquad (12)$$

where:

$T_0 = \epsilon$

$T_1 = ((\langle\cdot\ell\cdot\rangle \rightarrow 101)(\langle\cdot\ell\cdot\rangle \rightarrow 1)(\langle\cdot\ell\cdot\rangle \rightarrow 0)$

$T_2 = ((\langle\cdot n\cdot\rangle \rightarrow 101)(\langle\cdot n\cdot\rangle \rightarrow 1)(\langle\cdot n\cdot\rangle \rightarrow 0)$

$T_3 = ((\langle\cdot\ell\cdot\rangle \rightarrow 010)(\langle\cdot n\cdot\rangle \rightarrow 101)(\langle\cdot\ell\cdot\rangle \rightarrow 1)(\langle\cdot\ell\cdot\rangle \rightarrow 0)$

and where ⟨·ℓ·⟩ stands for ⟨·local·⟩ and ⟨·n·⟩ stands for ⟨·nonlocal·⟩.

Recognizers for table-languages

Much work has been done on the problem of designing syntax-directed recognizers[14] for languages defined by BNF grammars. This section will indicate that existing syntax-directed techniques for BNF can be extended to table-languages.

Recognition-operators

A syntax-directed recognizer needs a means of carrying out each event in the recognition sequence which corresponds to its dual in the generation sequence. For a table grammar this requires that a recognition-operator be defined for each grammar-operator. While the same names will be used for both types of operators, the context will make clear which type of operator is intended. An additional recognition-operator, *copy*, is also provided to allow a limited form of deterministic recognition. The recognition-operators have access to a *workspace* which is placed on the top of the table-tape. The *recognition*-operators are defined as follows:

copy:

This monadic operator has as its operand a terminal character from a set of *checkout-characters*. The set of checkout-characters is fixed for a particular grammar. "*copy* c" reads characters from the input and places then in the workspace until one of the checkout-characters is reached. The operation succeeds if this character is "c" otherwise it fails. If the operation succeeds, the control is passed along this path with the workspace initialized for use by a subsequent table-operation. If the operation fails the workspace is erased and recognition must proceed along an alternate path.

create, retrieve, recreate:

These recognizer-operators utilize the *workspace* as their identifier input. If these operators succeed, they erase the workspace in addition to their usual side effects in the active-table. Since these three recognition-operators require that the identifier be already present in the *workspace*, they must be preceded by the execution of *copy* which initializes the *workspace*.

illegal, replaces, copytable, erasetable:

These four recognition-operators are identical to their duals in a grammar. They neither utilize nor affect the contents of the workspace.

A summary of the definitions for the seven operators for a grammar and the eight operators for a recognizer are given in Appendices 1 and 2 respectively.

A recognizer for MAL programs

The table-grammar for MAL can be revised to place it in right-linear form since self-embedding recursion is not employed. In this form a recognizer or generator would not use the pushdown store and would only need the finite control and the table-tape. A deterministic recognizer for MAL programs is shown in Figure 2. A *copy* with reference to the respective delimiter has been inserted preceding each instance of *create, recreate* and *retrieve.* In two cases table-operations provide alternative paths from a node. However in each situation only one of the possible alternatives can succeed.

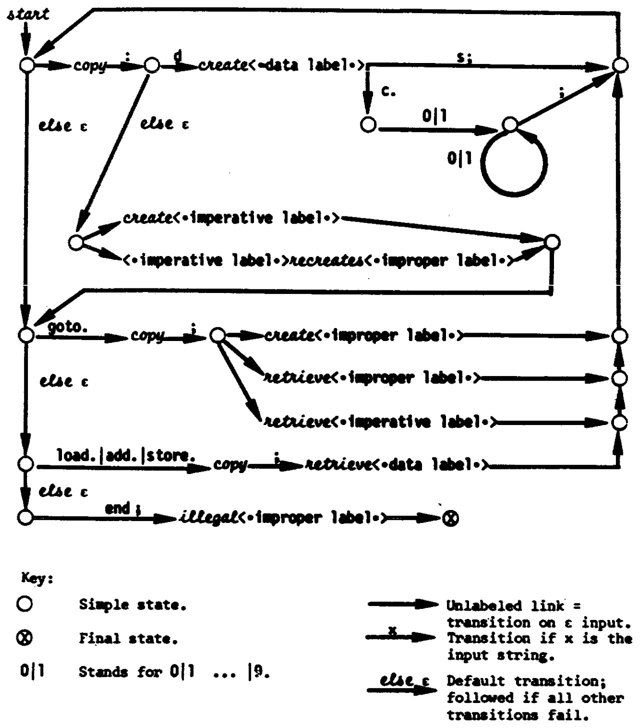

start
copy
:
d
create<·data label·>
s;
c.
;
0|1
0|1
else ε
else ε
create<·imperative label·>
<·imperative label·>recreates<·improper label·>
goto.
copy
:
create<·improper label·>
retrieve<·improper label·>
retrieve<·imperative label·>
else ε
load.|add.|store.
copy
:
retrieve<·data label·>
else ε
end ;
illegal<·improper label·>
Key:
Simple state.
Final state.
0|1 Stands for 0|1 ... |9.
Unlabeled link = transition on ε input.
x Transition if x is the input string.
else ε Default transition; followed if all other transitions fail.

Figure 2—A deterministic table recognizer for MAL programs

Consider the three alternatives which occur following the path for goto:

create ⟨·improper label·⟩ (1)

retrieve ⟨·improper label·⟩ (2)

retrieve ⟨·imperative label·⟩ (3)

If the identifier in the workspace is not in the table then (1) will succeed, and (2) and (3) will fail. If the identifier in the workspace is in the table, (2) will succeed if the selected table-category is ⟨·improper label·⟩, while (3) succeeds if the selected table-category is ⟨·imperative label·⟩. In each case there is only one path from the node which can succeed. If none succeed the device is blocked and the input-string is rejected.

Languages with self-embedding recursion cannot be placed in right-linear form and would require the use of the pushdown store as part of the recognition scheme. Also not all table-languages will be deterministically recognizable using the *copy* operator with its checkout-character set and workspace.

Regular table expressions

A *regular table expression* is an extension of those regular expressions where operators are limited to union, catenation and closure. Limited regular expressions, when embedded within a context-free grammar, have been shown[15] to be useful both in the representation of the syntax of programming languages and in the construction of processors which automatically create efficient recognizers. This section will indicate that the use of regular expressions within a rule of a grammar can be extended to the domain of regular table expressions.

Union, catenation, closure & distribution

A regular table expression is defined recursively as follows:

Let "a" be any terminal, ϵ the empty-string, and ϕ the null-set, then:

1. Any element in the set {a, ϵ, ϕ, *create* ⟨·i·⟩, *retrieve* ⟨·r·⟩, *illegal* ⟨·r·⟩, ⟨·i·⟩*recreates* ⟨·r·⟩, ⟨·i·⟩ *replaces* ⟨·r·⟩} is a regular table expression.
2. If ρ is a regular table expression, then so are: *copytable* ρ *erasetable* and $(\rho)^*$.
3. If ρ_1 and ρ_2 are regular table expressions defined for the same set of identifiers and having the same table-categories, then so are: $\rho_1\rho_2$ and $\rho_1 \cup \rho_2$.
4. No string is a regular table expression unless its being so follows from 1 to 3 above.

It is claimed without giving the proof that regular table expressions are closed under union, catenation and Kleene closure. However it is necessary to distinguish between the form of an expression and the set it defines. If L_1 is the language defined by ρ_1 and L_2 by ρ_2 and $L_3 = L_1L_2$ it does not follow that $L_3 = \rho_1\rho_2$. If L is the language defined by ρ, then it does not follow that $L^* = (\rho)^*$. This is due to the fact that various table-operators record information in the active-table so that generations to the right of a table-operator may not generate the same strings as they would if they were on its left. The definition of $(\rho)^*$ is $\bigcup_{i=0}^{\infty} \rho^i$ where $\rho^0 = \epsilon$ and $\rho^{i+1} = \rho\rho^i$.

The laws of distribution are obtained by definition. Let $\rho, \rho_1, \ldots, \rho_n$ be regular table expressions, then:

$$\rho(\rho_1 \cup \rho_2 \cup \ldots \cup \rho_n) = \rho\rho_1 \cup \rho\rho_2 \cup \ldots \cup \rho\rho_n ,$$

and

$$(\rho_1 \cup \rho_2 \cup \ldots \cup \rho_n)\rho = \rho_1\rho \cup \rho_2\rho \cup \ldots \cup \rho_n\rho .$$

Replacing recursion by the closure operator

In clause 1 of the definition of a regular table expression (given above), if the domain of "a" is extended to include grammar-categories, then two important identities can be shown to preserve the generative power of table-grammars (the proof itself is not given).

Let ρ_1, ρ_2 be regular table expressions that have no instance of $\langle a \rangle$ within them; further let "(" and ")" be grouping brackets for expressions and let them be disjoint from the set of terminals, then:

1. The rule $\langle a \rangle ::= \rho_1 \mid (\rho_2)\langle a \rangle$ can be written $\langle a \rangle ::= (\rho_2)^*\rho_1$.
2. The rule $\langle a \rangle ::= p_1 \mid \langle a \rangle(\rho_2)$ can be written $\langle a \rangle ::= \rho_1(\rho_2)^*$.

In both cases, $\langle a \rangle$ is now defined without the use of explicit recursion. By the use of a uniform substitution for $\langle a \rangle$, it can be eliminated from the set of rules of a table-grammar.

Extensions and limitations

The following extensions of table-grammars can be readily achieved within the framework given here. Some limitations are cited which lie outside the present model. Additional extensions which can overcome these limitations may be possible.

A partitioned identifier set

For some languages, the identifier set is partitioned by the definitions of the language. In ASA Fortran, if an identifier has no explicit declaration the type is implied by the rule: "names beginning with letters I to N are type integer, all others are type real." This rule partitions the undeclared identifiers into two disjoint subsets. To handle this case *create* would be replaced by two operators *create-integer* and *create-real.* For a particular grammar these new operators would be defined to select undeclared identifiers from their respective disjoint partition of the total set of identifiers.

Use of an implicit retrieve operator

In a very large grammar, it may be desirable to omit the operator *retrieve* and let the table-category itself, appearing alone in the grammar, be in effect an implicit call, in which the effective presence of *retrieve* is to be understood. The generative power of the notation with *retrieve* omitted would be identitical to that where *retrieve* is explicitly used.

Limitations

A table-grammar, as defined here, is able to specify the declarations of scalar variables and labels. It is unable to specify all the restraints required for subscripted variables and for procedure calls when argument lists are used. In addition, table-grammars are limited to declarations where the identifier set is static. The IMPLICIT statement of Fortran sets up dynamic partitions of the identifier set. Even the extension of *create* given above is able to handle only static partitions of the identifier set.

ACKNOWLEDGMENTS

The author wishes to thank R. A. Stone ot Western Electric, Princeton, C. B. Jones of I.B.M. Vienna, C. N. Mooers of Rockford Research, Cambridge, Mass., and M. A. Harrison of the University of California, Berkeley, each of whom contributed constructive suggestions which have been included in the paper.

REFERENCES

1 P NAUR
Documentation problems: ALGOL 60
Communications of the ACM Vol 6 No 3 March 1963 p 78

2 P NAUR
Revised report on the algorithmic language ALGOL 60
Communications of the ACM Vol 6 No 1 January 1963 pp 1–17

3 N CHOMSKY
On certain formal properties of grammars
Information and Control Vol 2 1959 pp 137–167

4 S GINSBURG
The mathematical theory of context-free languages
McGraw-Hill 1966

5 R W FLOYD
On the nonexistence of a phrase structure grammar for ALGOL-60
Communications of the ACM Vol 5 No 9 September 1962 pp 483–484

6 K IVERSON
A method of syntax specification
Communications of the ACM Vol 7 No 10 October 1964 pp 588–589

7 J W CARR J WEILAND
A nonrecursive method of syntax specification
Communications of the ACM Vol 9 No 4 April 1966 pp 267–269

8 F G DUNCAN
Notational abbreviations applied to the syntax of Algol
SICPLAN Notices ACM Vol 2 No 11 November 1967 pp 28–43

9 K BANDAT
On the formal definition of PL/I
Proc S J C C Vol 32 1968 pp 363–374

10 S GINSBURG S A GREIBACH M A HARRISON
Stack automata and compiling
Journal of the ACM Vol 14 No 1 January 1967 pp 172–201

11 G WHITNEY
The generation and recognition properties of table languages
IFIP Congress Software I August 1968 pp B18–B22

12 G WHITNEY
The position of table languages within the hierarchy of nondeterministic on-line tape bounded during machine languages
IEEE Conference Record Ninth Annual Symposium on Switching and Automata Theory October 1968 pp 120–130

13 R E STERNS P M LEWIS
Property languages and table machines
Ibid pp 106–119
14 J FELDMAN D GRIES
Translator writing systems
Communications of the ACM Vol 11 No 2 February 1968
pp 77–113
15 V TIXIER
Recursive functions of regular expressions in language analysis
Stanford University Computer Science Dept Report CS58
March 1967

APPENDIX I

Table-expression	Conditions	Value-returned	Side-effects
create ⟨·i·⟩	Selected-identifier must be undeclared	Selected-identifier	Add to the active-table the rule: ⟨·i·⟩ → selected-identifier
retrieve ⟨·r·⟩	There must exist within the active-table a rule: ⟨·r·⟩ → identifier	Selected-identifier	None
⟨·i·⟩ *recreates* ⟨·r·⟩	There must exist within the active-table a rule: ⟨·r·⟩ → identifier	Selected-identifier	The selected rule is changed to: ⟨·i·⟩ → identifier
⟨·i·⟩ *replaces* ⟨·r·⟩	None	ϵ	Within the active-table each ⟨·r·⟩ is replaced by ⟨·i·⟩
illegal ⟨·r·⟩	The operation is blocked if ⟨·r·⟩ is present in the active-table	ϵ	None
copytable	None	ϵ	A duplicate copy of the active-table is placed on the top of the table-tape
erasetable	None	ϵ	The active-table is erased

Note: ⟨·i·⟩ stands for an incident table-category and ⟨·r·⟩ for a resident table-category. ϵ is the empty-string.

SUMMARY OF DEFINITIONS FOR GENERATION-OPERATORS

APPENDIX II

Table-expression	Conditions	Side-effects if operation succeeds: In active-table	Side-effects if operation succeeds: In workspace
copy c	Succeeds if the next check-out character is c	None	Characters up to c are copied into the workspace
create ⟨·i·⟩	Identifier in the workspace must be undeclared	Add the rule: ⟨·i·⟩ $\rightarrow$ identifier	Erased
retrieve ⟨·r·⟩	There must exist within the active-table a rule: ⟨·r·⟩ $\rightarrow$ identifier such that identifier = contents of workspace	None	Erased
⟨·i·⟩*recreates* ⟨·r·⟩	Same as for *retrieve* ⟨·r·⟩	The selected rule is changed to: ⟨·i·⟩ $\rightarrow$ identifier	Erased

Note: a. the execution of *copy* must precede evaluation of expressions containing *create*, *retrieve*, or *recreate*.
b. the definitions for *illegal*, *replaces*, *copytable* and *erasetable* are the same as given in Appendix I.

SUMMARY OF DEFINITIONS FOR RECOGNITION-OPERATORS

A hierarchical graph model of the semantics of programs

by TERRENCE W. PRATT

The University of Texas
Austin, Texas

INTRODUCTION

The problem of developing an adequate formal model for the semantics of programming languages has been under intensive study in recent years. Unlike the area of syntax specification, where adequate models have existed for some time, the area of semantic specification is still in the formative stages. Development of formal semantic models has proceeded along two main lines, lambda-calculus models (e.g., Landin,[1] Strachey[2]) and directed graph models (e.g., Narasimhan,[3] Floyd[4]). This paper describes a model for the semantics of programs based on hierarchies of directed graphs.

A formal model or theory of the semantics of programming languages must provide descriptions on a number of different levels, much as the theory of context-free grammars provides descriptions of the syntax of programming languages on a number of different levels. At the most general level a semantic theory provides a framework for describing a class of programming languages and investigating the mathematical properties of their formal representations in the model. At this general level the context-free grammar model of syntax has been particularly successful, giving rise to an extensive mathematical theory as well as providing characterizations of specialized syntactically-similar classes of languages. One would hope that adequate models of semantics would lead to the same sort of mathematical development and to the classification of semantically-similar programming languages.

At a more specific level, a semantic model must provide descriptions of the semantics of particular programming languages, much as a particular context-free grammar may be used to describe the syntax of a particular programming language. One would expect the formal specification of the semantics of a programming language to allow conciseness and unambiguity in definition and also to lead to definition of new languages which are more "regular" semantically.

At a still more specific level, a semantic model must provide descriptions of particular programs, and these descriptions must be constructable from the representation of the program (the syntax) in a straightforward manner. Thus it should be possible to construct a "compiler" which will translate a program in its written representation into its equivalent in the semantic model. From this representation the formal properties of the model may be used to guide further processing, by producing, for example, a more efficient program which is semantically equivalent or by translating the program into another representation, such as a semantically equivalent program in another language.

At the level of complete specification, a semantic model must provide a representation of the data used by a program and allow execution of the program on the particular data. Thus it should be possible to construct an "interpreter" which will execute a program-data pair in its representation in the model.

The importance of the development of adequate models of semantics stems from the probable gains to be realized by their use, both in the area of definition of programming languages, where the formal definition of semantics should lead to semantically unambiguous as well as more easily intelligible languages, and in the area of processor construction, where it is likely that a better understanding of the semantic features of languages will lead to more powerful and efficient processing techniques for those features.

The model for semantics described in the following sections is based on the use of hierarchies of directed graphs to represent both programs and data. The hierarchical graph or H-graph concept which is basic to the model is defined in the next section, along with the concepts of "level" and "path" which serve as important structural concepts in the application of H-graphs to the description of programming languages. In a later section the basic semantic model is introduced by structuring further the "atomic units" which lie at the lowest level of the hierarchy in H-graphs. Basic programming concepts such as "program," "data," and "execution of a program" are defined also. In the last section examples are given to illustrate informally how a number of semantic features of actual programming languages may be represented in a natural way in the model. Included are complete models of the semantics of particular Turing machine, Lisp, and Fortran programs. Finally, a concluding section attempts to evaluate the model and view possible applications and extensions.

Hierarchical graphs (H-graphs)

In this section a generalization of the mathematical concept of "finite directed graph" is presented which forms the basis for the model of semantics presented in the next section.

Basically, a directed graph (sometimes called a "network") is composed of a finite set of "nodes," and a finite set of "edges" connecting pairs of nodes in a "one-way" manner. Commonly, nodes are represented as circles or boxes and edges as arrows connecting nodes. Examples of the use of directed graphs in computer science include flow charts, automata transition diagrams, PERT networks, and representations of list structures and trees.

Three extensions of the basic concept of directed graph are of interest here. The first two lead to a definition of "extended directed graph," and the third to a definition of "hierarchical directed graph."

Extended directed graphs

Informally, an extended directed graph is a directed graph with a designated node (called the "entry point") and with the edges leaving each node uniquely labeled. Formally:

Defn: An *extended directed graph* is an ordered quadruple (N, L, S, E) where N is a finite non-empty set (of *nodes*),

L is a finite set (of *labels*),

S is an element of N (called the *entry point*), and

E is a partial function from N x L into N, defining the *edges*. If E(n, p) = q, then there is said to be an edge from node n to node q with label p. Note that there may not be two edges leaving a node with the same label, but two edges leaving a node with different labels may end at the same node; thus "parallel" edges are allowed. Ignoring the contents of the nodes, every flow chart may be considered as an extended directed graph.

H-graphs

A hierarchy is introduced into a set of extended directed graphs (hereafter called simply "graphs") in the following manner. A universe U of atomic units (distinct symbols) is assumed. A hierarchical graph or H-graph over U is (informally) a graph in which the nodes are considered to be "containers," each of whose contents is either an atomic unit from U or an H-graph over U. Thus in flow-chart terms, the analogue of an H-graph is a flow chart in which each box contains either an "atom" or another flow chart, whose boxes may in turn contain atoms or flow charts, and so forth, to any depth. Formally:

Defn: If U is a set (of *atomic units* or *atoms*), then an *H-graph* over U is an ordered pair (N, V) where:

1. N is a finite set (of *nodes* or *containers*).
2. V (the *contents* or *value function*) is a function mapping N into U $\cup$ {x:x is a graph with nodes from N and labels from U}. If c ϵ N, then V(c) is called the *contents* or *value* of c.

Levels

The hierarchical structure of an H-graph is brought out by considering the levels of its nodes.

Defn: The *level of a node* c is defined recursively as:

a. 1 if V(c) ϵ U (i.e., if c contains an atomic unit).
b. n if V(c) is a graph (N, L, S, E), the level of each node in N is in the set {1, 2, . . ., n − 1}, and at least one node in N has level n − 1.

If a node is not of level n for any n, then it is termed a *recursive node*.

Defn: The *node set* of a node c in an H-graph is defined to be:

a. if c has level 1, then Λ, the empty set,

or

b. if c contains a graph (N, L, S, E), then N $\cup$ [$\bigcup_{x \epsilon N}$ {node set of x}].

Thus the node set of a node is composed of all the nodes "reachable" in the hierarchy starting from that node.

Defn: An H-graph (N, V) is *recursive* if N contains a recursive node.

Defn: The *atom set* of a node c is the set of values of all level 1 nodes of the node set of c. Thus the atom set of a node is the set of atomic units "reachable" in the hierarchy starting from that node.

Paths

Along with the concept of level, the idea of a "path" in an H-graph is important in the development of the model of the next section. The concept of path will be defined first for graphs and then extended to H-graphs. Informally, a path in a graph from one node to another is just a set of edges which may be traversed in going from the first node to the second, passing from node to node along connecting edges. Formally,

Defn: If G = (N, L, S, E) is a graph, then a *path* from node x to node y of G is an ordered sequence $(x, k_0, k_1, \ldots, k_m)$ where each k_i is a label such that $E(x, k_0) = n_1$, $E(n_1, k_1) = n_2, \ldots$, and $E(n_m, k_m) = y$. In general, certain nodes of a graph will have no edges leaving them. Of particular importance here are the paths which begin at the entry point of a graph and terminate at such nodes.

Defn: A *path beginning at node x* is a path from node x to a node y such that y has no edges leaving it, i.e., such that E(y, k) is not defined for any $k \in L$.

Defn: A *path in a graph* G is any path beginning at s, where s is the entry point of G.

Extending this concept to H-graphs, if C is a node of an H-graph, then (informally) a path through C is composed by taking a path through the graph contained in C, a path through each of the graphs contained in the nodes encountered in that path, and so forth down the hierarchy until level 1 nodes are reached. Formally,

Defn: If G is an H-graph and C is a node of G, then a *path* P_c *through node C* is represented by:

1. if C contains an atomic unit A, then (C, A)

or

2. if C contains a graph, then (C, P) where $P = (P_s, e_0, p_{n_0}, e_1, \ldots, e_m, P_{n_m})$ and $(s, e_0, \ldots, e_m)$ is a path in G passing through nodes $s, n_0, \ldots, n_m$ and each P_{n_i} is a path through node n_i.

Defn: The *atom sequence* of a path is simply the sequence of atomic units encountered in following the path. In the representation given above, the atom sequence is simply the sequence of atomic units (not including edge labels) obtained in scanning the representation of a path from left-to-right ignoring everything except atomic units.

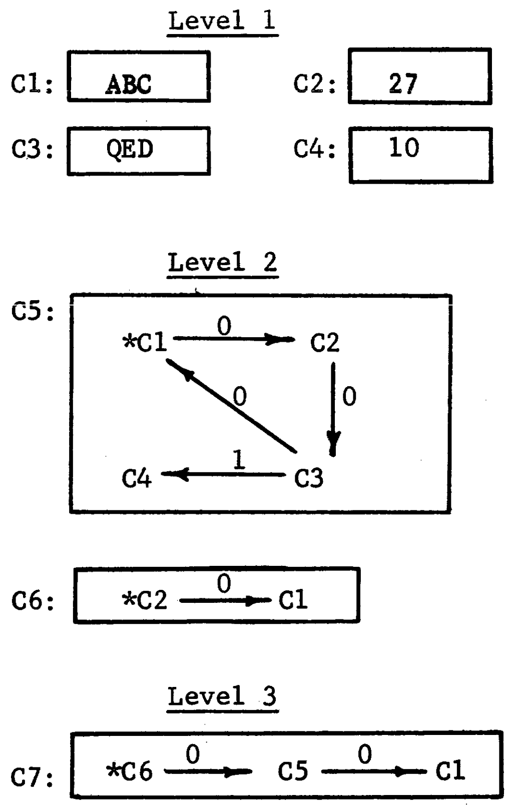


Figure 1—Three level H-graph

For example, Figure 1 represents an H-graph, where C1, C2, . . ., C7 are nodes each of whose contents is given by the contents of the correspondingly labeled box. One path through node C7 is represented by: (C7, ((C6, ((C2, 27), 0, (C1, ABC))), 0, (C5, ((C1, ABC), 0, (C2, 27), 0, (C3, QED), 1, (C4, 10))), 0, (C1, ABC))).

The atom sequence for this path is: (27, ABC, ABC, 27, QED, 10, ABC).

Further development of the theory of H-graphs is outside the scope of this paper. This brief introduction will suffice for the purposes here.

The basic semantic model

On the basis of the H-graph concept, an elementary model of semantics is developed in this section, including definitions of program, data, and execution of a program.

Program and data H-graphs

The semantic model is based on H-graphs with a more highly structured universe of atomic units. For the semantic model, the universe U of atomic units is assumed to have the following structure:

$U = D \cup \phi$ (D and ϕ not necessarily disjoint) where D is a set of *data* units and ϕ is a set of *operator instances.*

$D = D_1 \cup D_2 \cup \ldots \cup D_n$ (the D_i's not necessarily disjoint) where each D_i is a set of data units forming a *data type class.*

(D_i = e.g., integers, reals, bits, characters, strings)

Similarly:

$\phi = \phi_1, \cup \phi_2 \cup \ldots \cup \phi_m$ (ϕ_i's not necessarily disjoint) where each ϕ_i is a set of operator instances forming an *operator type class.*

(ϕ_i = e.g., "add," "multiply," "popup," "differentiate")

Definitions:

1. An *atomic unit* $a \epsilon U$ is said to be *of type T* (where T is a data or operator type class) if $a \epsilon T$. Since type classes are not necessarily disjoint, the type of an atomic unit is not necessarily unique.
2. A *node n* of an H-graph over U is said to be *of type T* if the atom set of n is a subset of type class T.
3. An *H-graph* is *of type T* if each of its nodes is of type T.
4. A *data node* is a node whose atom set is a subset of D.
5. A *data H-graph* is an H-graph whose nodes are all data nodes.
6. A *program node* is a node whose atom set is a subset of ϕ.
7. A *program H-graph* is an H-graph whose nodes are all program nodes.

Operator instances

Although the concept of data type class is familiar from actual programming languages, that of operator instance is not as common.

Defn: An *operator instance* Q is an ordered triple Q = (I, 0, f), usually written f(I; 0), where I is a finite set of (*input*) nodes, 0 is a finite set of (*output*) nodes, and f (the *operator*) is a function from X into Y, where X and Y are sets of H-graphs, each H-graph in X and Y containing the nodes in I and 0.

A typical example would be the operator instance

"({A, B}, {C}, add)" or "add (A, B; C)"

with input nodes A and B, output node C, and operator "add" which maps any H-graph containing the nodes A, B, and C, and in which A and B have numerical values, into the H-graph identical to the original except with C containing the sum of the values of A and B.

Execution of an operator instance is a primitive concept by which is meant the application of the operator to its arguments found in its input nodes, producing values found in its output nodes. When an operator instance is executed, it is understood that the H-graphs defined by its input nodes completely define its arguments, in the sense that given the same input H-graphs, the same values will be produced as output. Similarly, the effect of an operator is assumed to be localized in the H-graphs produced as the values of its output nodes, in the sense that if the value of a node is changed by the operator then that node must be in the node set of one of the output nodes. Thus the set of input nodes of an operator instance in a sense specifies the maximum set of data structures which may influence the effect of that operator instance, and similarly, the output nodes specify the maximum extent of the effects produced.

Note that only the input and output nodes of an operator instance are fixed, not the contents of these nodes, so that different executions of the same operator instance will, in general, produce different results, even though the input and output nodes are the same. Only if the H-graphs defined by the input nodes are identical will execution necessarily produce the same results in the output nodes.

Execution of a program H-graph

Any program node of an H-graph defines a set of possible execution sequences of operator instances as follows:

Defn: If c is a program node, then the sequence $p_1, p_2, \ldots, p_n$ of operator instances is an *execution sequence* of c if $(p_1, p_2, \ldots, p_n)$ is the atom sequence of some path through c.

Every path through a program node defines a possible execution sequence of operator instances. Execution of a program node involves the choice of a path through that node and execution of the operator instances in order from the execution sequence defined by that path. Corresponding to actual execution of programs in a programming language, the process of choosing a path may be represented as a process which in a sense actually traces out a path step by step, executing operator instances as they are encountered in the path, and choosing the next step of the path as a dynamic function of the results of execution of the previous operator instances encountered. Thus the choice of execution sequence is determined by the "data," that is by the H-graphs in the input node set of the operator instance being executed. The process of execution of a *program node* c of an H-graph (N, V) may be defined precisely as follows by use of a designated level 1 node P of type "label:"

1. If c is a level 1 node, then execute the operator instance contained in c.
2. If c is not a level 1 node and contains the graph (N, L, S, E), then:
   a. Set the current node = S.
   b. Execute the current node (saving the name of the node being executed in a stack until execution is complete).
   c. If there is no edge leaving the current node, then stop (returning to the next higher level to continue execution if necessary).
   d. If there is exactly one edge (labeled k) leaving the current node, then set current node = E(current node, k) and go to (b).
   e. If there is more than one edge leaving the current node, then set the current node = E(current node, V(P)) and go to (b).

Note that at a branch in a program graph the choice of which branch to follow is determined by the label contained in the designated node P. Since P may be one of the output nodes of an operator instance, execution of an operator instance may change the value of P, and thus the "flow of control" is determined dynamically by the operator instances and the "data." Clearly, a path through node c is determined by this process if the process terminates.

Correspondence between the model and actual programming languages

The general model of the previous section serves as a framework for the description, comparison, and classification of the semantics of different programming languages. A model of the semantics of a particular programming language is formed by specification of a particular universe of atomic units together with a set of restrictions or constraints on the types of data and program H-graphs which may be constructed on this universe. A model of a program or a data structure in the language then is a program or data H-graph on the given universe which satisfies the constraints of the language. The classification and comparison of the semantics of languages is based on the classification and comparison of the properties of the universes of atomic units and of the properties of the constraints on H-graphs. Although detailed descriptions of actual programming languages are outside the scope of this paper, an attempt is made in this section to develop in a general way a correspondence between features of actual programming languages and properties of the model.

Particular features of actual programming languages may commonly be represented in more than one way in the model. The examples given below indicate only one possible way that particular constructs may be represented, without precluding the possibility that other representations may be possible and even desirable in certain models of an entire language.

In the examples, nodes are represented by ovals or polygons and edges by arrows. All nodes and edges are labeled. The contents of a node will ordinarily be written inside the oval or rectangle representing it. The entry point node of a graph is indicated by an * next to the node. Thus, for example, Figure 2 represents a node C whose value is a graph on nodes C1, C2, and C3, with entry point C1 and E(C1, k) = C2, E(C2, k) = C3, E(C3, k) = C1. The nodes C1, C2, and C3 have as values the atoms A, B, and C, respectively.

In cases where node names and/or edge labels are not significant, they will ordinarily be omitted. Thus the same graph might be written alternatively as in Figure 3. Operator instances will be represented in the form:

operator name ($i_1 i_2, \ldots, i_n; 0_1, 0_2, \ldots, 0_m$)

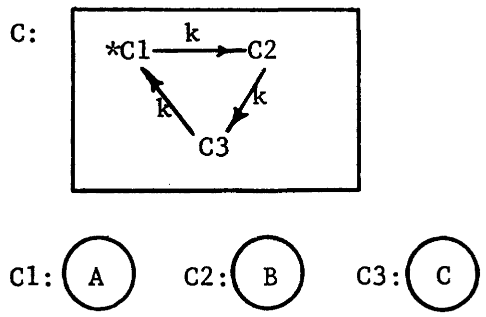


Figure 2

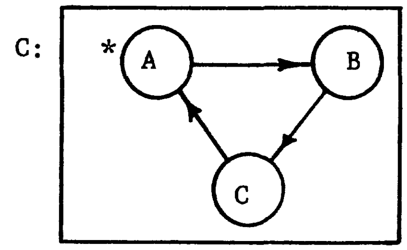


Figure 3

where each i_k is the name of an input node and each 0_k is the name of an output node.

Three complete models of programs and data are given in this section, for Turing machines, LISP, and Fortran. Although no attempt is made to provide complete descriptions of the languages, the models of the particular programs used are constructed so as to indicate how a general model might be constructed.

Turing machine semantics. Consider first the representation of the semantics of a particular Turing machine. Intuitively, a Turing machine is very simple semantically. Both the program (the functional matrix) and the data (the tape) are of simple structure. This makes a model of the semantics of Turing machines a useful test case for a semantic theory. The general class of Turing machines that will be modeled have a single tape and read head, and at each move they will (1) read the symbol in the square being scanned, (2) change state, (3) write a symbol, and (4) move left or right one square.

a. *Atomic units.* The universe of atomic units U for Turing machines over a particular alphabet A is composed of:

$$U = A' \cup \phi$$

where A' and ϕ are disjoint, $A' = A \cup \{\text{left, right}\}$, A is the alphabet of the Turing machine, {left, right} are labels, and ϕ = READ $\cup$ WRITE $\cup$ MOVELEFT $\cup$ MOVERIGHT $\cup$ NO-OP where READ, WRITE, MOVELEFT, MOVERIGHT, and NO-OP are the names of disjoint operator type classes defined below.

b. *Data structures.*
Level 1 nodes. Except for the special node P and the nodes containing the alphabet symbols, all level 1 nodes correspond to tape squares containing a symbol from the alphabet A. P is the node mentioned in the execution algorithm for program H-graphs, and for each alphabet symbol a node is required containing that symbol (for the WRITE operator).
Level 2 nodes. Only two nodes are needed, TAPE and HEAD. TAPE contains a two-way list (of level 1 nodes) with edges labeled "left" and "right." HEAD contains a single level 1 node which corresponds to the tape square being scanned.

c. *Program structures.*
Level 1 nodes. Contain single operator instances.
Level 2 nodes. Only one node PROG is needed, which contains a graph representing the functional matrix of the Turing machine.

d. *Operator type classes.* See Table I. (H = HEAD, T = TAPE, V is the function which, given a node, returns the value of that node.)

e. *Example.* For the 2-state 3-symbol Turing machine given in Figure 4 an H-graph representation is given in Figure 5.

Execution of the program H-graph PRØG simulates the operation of the Turing machine. Execution of PRØG involves simply following a path from the entry point (node P9) of the graph in PROG to the (unique) node P5 of the graph which has no exiting edges. This path is chosen according to the algorithm of the

Table I—Turing machine operator instances

Class	Form of Operator Instances	Function
READ	READ(H; P)	Transfers V(V(H)) to P (i.e., copies the symbol being scanned into P)
WRITE	WRITE(α, H; H)	Sets the value of the node V(H) = V(α)
MOVELEFT	MOVELEFT(H, T; H, T)	Let α = V(H), set V(H) = E(α, left) where E is the edge function of the graph in T. If E(α, left) is not defined, create a new node with a unique name β. Set V(H) = β. Define E(α, left) = β and E(β, right) = α and set V(β) = $\not{b}$ (empty tape square symbol of A)
MOVERIGHT	MOVERIGHT(H, T; H, T)	Same definition as MOVELEFT, replacing "left" by "right" and "right" by "left."
NO-OP	NO-OP()	The identity operation (does nothing)

MATRIX

Symbol / State	0	1	#
S_0	1,R,S_0	0,R,S_0	#,L,S_1
S_1	1,L,S_1	0,L,S_1	Halt

INPUT TAPE

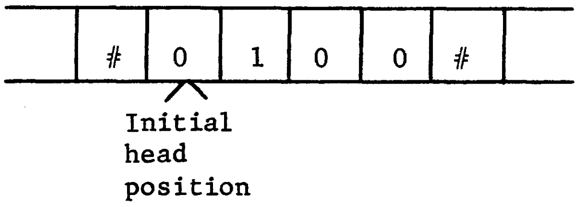


Figure 4—Turing machine initial configuration

preceding section. Thus beginning at P9, the operator instance in P9 is executed (corresponding to reading the tape square under the read head and storing the symbol read in node P). Then the edge leaving P9 is chosen which is labeled with the symbol contained in P (in this case 0). This edge leads to P0, which becomes the current node. The operator instance in P0 is executed (writing a 1 on the tape square under the read head). The single edge leaving P0 is followed to P3, the operator instance contained in P3 is executed, etc. Execution continues in this manner until node P5 is reached.

In the Turing machine representation both program and data have only two levels, giving a hierarchically simple structure. The given representation readily extends to arbitrary Turing machines and initial configurations.

Lisp semantics. As a second example, consider the problem of representing the semantics of the following Lisp function with its argument:

(LABEL(LAST(LAMBDA(X) (CØND

((NULL(CDR X)) (CAR X))

(T (LAST (CDR X))))))) ((A (B C) D E))

An H-graph model may be constructed for this function in such a way that extension of the model to the remainder of the Lisp language is possible.

In representing the Lisp function two major problems arise:

1. The sequence of operations specified implicitly through function composition and the order of evaluation conventions of Lisp must be made explicit in the H-graph representation.
2. The pairing of formal parameters and actual parameters, and the transmission of evaluated arguments to the functions using them must be handled by explicit use of versions of the standard Lisp interpreter A-list and pushdown list.[5] This is due to the fact that the H-graph model contains no built-in provision for argument transmission to subprograms.

A representation of the semantics of the above Lisp function may be constructed as follows:

a. *Atomic units.* Let the universe U of atomic units be:

$$U = A \cup \{T, NIL\} \cup \emptyset$$

where A is the set of Lisp atoms and $\emptyset$ = CAR $\cup$ CDR $\cup$ NULL $\cup$ PAIR $\cup$ VALUE $\cup$ PØPUP where these data type classes are defined below.

b. *Data structures.* Lisp list structures are represented as hierarchical data graphs, where sublists are represented by lower levels in the hierarchy. A list of n elements is represented by a graph of n nodes, connected in sequence. If a list element is an atom, its corresponding graph node contains that atom as its value. If a list element is a sublist of m elements, its corresponding graph node contains a graph of m nodes representing the sublist. Thus the list:

(A, (B, C, D), ((E, F), G))

is represented by the H-graph in Figure 6. To handle argument transmission, certain auxiliary data structures will be necessary:

1. OP, a node which contains a list, used as a stack to hold operands;
2. ALIST, a node which contains a list, used as a stack to hold paired formal parameters and their values.

c. *Program structures.* Like all Lisp program structures not using PROG, the program graph for the above function is a tree. Since the function contains a recursive function call, the program graph is recursive.

d. *Operator type classes.* See Table II. (V is the

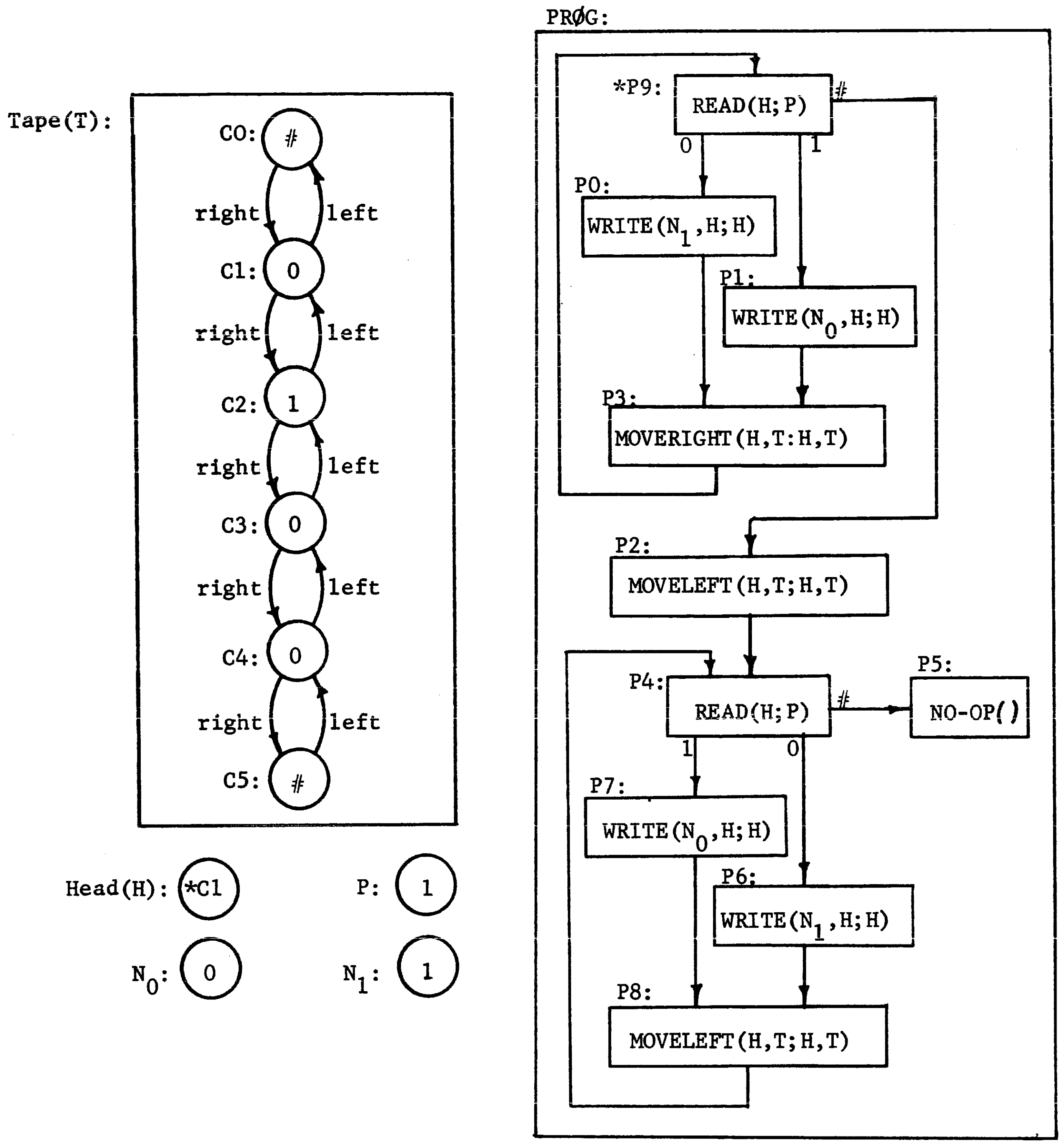


Figure 5—Turing machine H-graph representation

function which, given a node, returns the value of that node.)

e. *Example.* The Lisp function:

(LABEL(LAST(LAMBDA(X)(CØND

((NULL(CDR X)) (CAR X))

(T (LAST (CDR X))))))) ((A (B C) D E))

is represented by the H-graph of Figure 7.

Note that the sequence of operations performed during execution of the node LAST corresponds closely to the sequence executed by an ordinary Lisp interpreter: first the formal parameter, X, and the actual parameter, the list in C1, are paired on the ALIST, then X is evaluated, CDR of its value is taken, and

Table II—LISP operator instances

Class Name	Operator Instance Form	Function
CAR	CAR(ØP; ØP)	Let α = entry point of V(ØP). Let β = entry point of V(α). Set V(α) = V(β).
CDR	CDR(ØP; ØP)	Let α = entry point of V(ØP). Let β = entry point of V(α). Set V(α) = a graph obtained from V(α) by setting the entry point to E(β, ℓ) and deleting node β. If E(β, ℓ) is not defined, set V(α) = NIL.
NULL	NULL(ØP; ØP, P)	Let α = entry point of V(OP). Set V(α) = V(P) = T if V(α)′ = NIL and = NIL otherwise.
PAIR	PAIR(γ, ØP; ALIST, ØP)	Let α = entry point of V(ØP) and β = entry point of V(ALIST). (1) Pushdown V(ALIST), let C = new entry point (i.e., replace the graph in ALIST by $*C \rightarrow \beta \rightarrow \ldots$ where C is a new node and $*\beta \rightarrow \ldots$ was the old graph). (2) Set V(C) = the graph $*\gamma \rightarrow \alpha$. (3) Popup ØP (i.e., replace V(ØP) = the graph $*\alpha \rightarrow \alpha' \rightarrow \ldots$ by the graph $*\alpha' \rightarrow \ldots$
VALUE	VALUE(γ, ALIST; ØP)	(1) Find first node Q on list V(ALIST) such that γ = entry point of V(Q). Then V(Q) is a graph $*\gamma \rightarrow \alpha$. Let C be a new node. (2) Pushdown C onto list V(ØP) (i.e., replace $*\beta \rightarrow \ldots$ by $*C \rightarrow \beta \rightarrow \ldots$). (3) Set V(C) = V($\alpha$).
POPUP	POPUP(α; α)	Popup list V(α) (i.e., replace $*\beta \rightarrow \gamma \ldots$ by $*\gamma \rightarrow \ldots$).

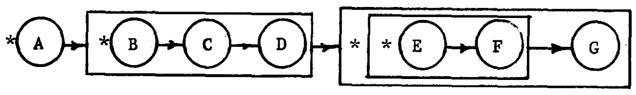


Figure 6—List structure H-graph representation

NULL of that value is taken. The value of NULL, T or NIL, is then used to control a branch. If T, the execution ends after the CAR of the value of X is taken. If NIL, then the CDR of the value of X is taken and execution descends a level at the recursive node N, which contains a graph of the single node LAST. The stack OP is used throughout to communicate results between operator instances. Execution of the above program H-graph LAST will result in the node ØP containing a graph whose entry point node contains the value of the function; thus ØP: $\boxed{* \text{E} \rightarrow \text{NIL}}$ is the final state of the node ØP, where E is the value of the function LAST for the given argument.

Fortran semantics. As a final example, consider the problem of representing the semantics of the following Fortran program:

```
   PRØGRAM EXAMPLE
   DIMENSIØN M(2, 5)
   DØ 2 I = 1, 5
   M(1, I) = I
 2 M(2, I) = IFACT(I)
   END
   FUNCTION IFACT(N)
   K = 1
```

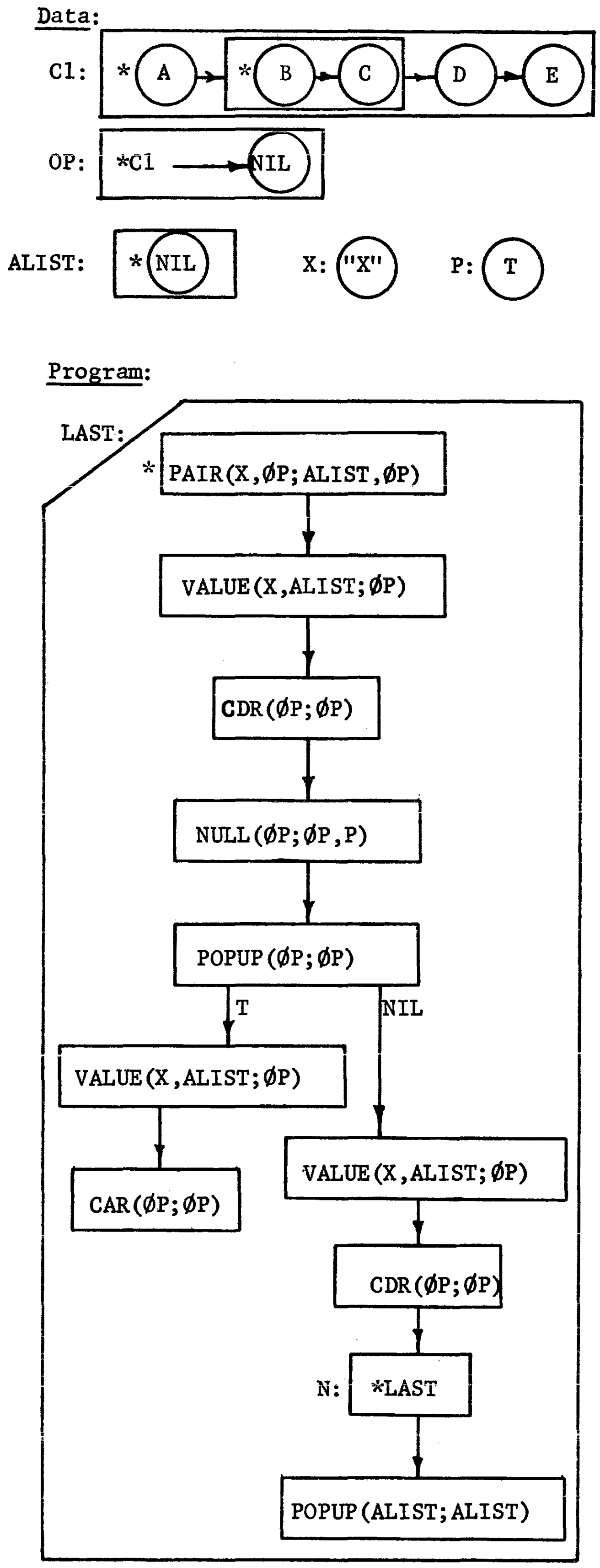


Figure 7—Lisp function H-graph representation

```
  DØ 2 J = 1, N
2 K = K*J
  IFACT = K
  RETURN
  END
```

As with Lisp, it is necessary to provide explicitly a mechanism for argument transmission to subprograms in the model of Fortran.

The H-graph representation may be constructed as follows:

a. *Atomic units.*

$$U = I \cup \{=, \neq\} \cup \emptyset$$

where I = set of integers

$\{\neq, \neq\}$ = set of labels

and

$\emptyset$ = the union of the operator type classes defined below.

b. *Data structures.* Simple variables are represented as level 1 data nodes. Two-dimensional arrays are represented by a graph (of level 1 nodes) with edges labeled 1 and 2 indicating rows and columns, respectively, as M in Figure 8.

c. *Program structures.* The program structures correspond roughly to flow charts of the Fortran programs. Subprograms are represented by separate levels in the hierarchy of program graphs.

d. *Operator type classes.* See Table III. (V = function which, given a node, returns the value of that node.)

e. *Example.* The program may be represented as the H-graph of Figure 8. Note that the loops implied by the two DØ statements are made explicit in the H-graph representation, and the operations of array referencing and argument transmission which are implicit in the Fortran program become explicit in the model as the operations REF2 and SETADDR-SET, respectively. Execution of the H-graph EXAMPLE results in the first row of the array M being filled with the integers 1–5 and the second row with the integers 1!–5!.

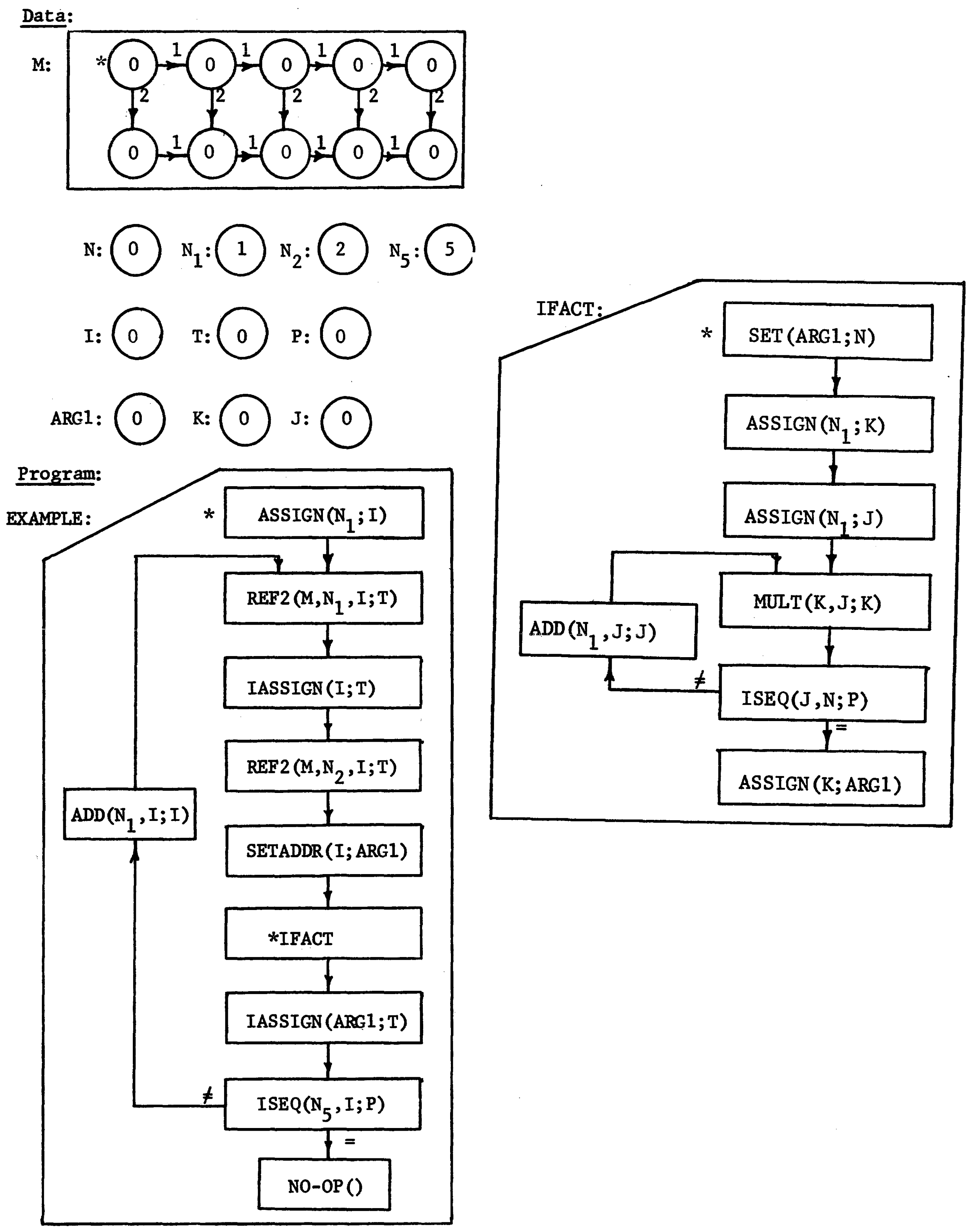


Figure 8—Fortran program H-graph representation

Table III—Fortran operator instances

Class Name	Operator Instance Form	Function
ADD	ADD$(\alpha, \beta; \gamma)$	Sets $V(\gamma) = V(\alpha) + V(\beta)$
MULTIPLY	MULTIPLY$(\alpha, \beta; \gamma)$	Sets $V(\gamma) = V(\alpha)*V(\beta)$
ASSIGN	ASSIGN$(\alpha; \beta)$	Sets $V(\beta) = V(\alpha)$
IASSIGN	IASSIGN$(\alpha; \beta)$	Sets $V(V(\beta)) = V(\alpha)$ (where $V(\beta)$ is a single node graph)
SETADDR	SETADDR$(\alpha; \beta)$	Sets $V(\beta) =$ the single node graph: $*\alpha$
SET	SET$(\alpha; \beta)$	Sets $V(\beta) = V(\alpha')$ where $\alpha' =$ entry point node of the graph $V(\alpha)$
ISEQ	ISEQ$(\alpha, \beta; P)$	Sets $V(P) = \begin{cases} = \text{ if } V(\alpha) = V(\beta) \\ \neq \text{ if } V(\alpha) \neq V(\beta) \end{cases}$
REF2	REF2$(\alpha, \beta, \gamma; \delta)$	Sets $V(\delta) =$ node corresponding to $\alpha(\beta, \gamma)$ in 2-dimensional array α
NO-OP	NO-OP( )	No-operation

CONCLUSION

In this paper the general outline of a hierarchical directed graph model of the semantics of programs has been sketched, and a number of examples of its use in modeling particular features of programming languages have been given. Many interesting questions remain concerning such models. Preliminary indications are that the model provides both mathematical tractability and intuitively appealing characterizations of the semantics of programming languages. It remains to be shown that these indications will remain valid as more extensive development is undertaken. At present, work is proceeding along two main lines. On the one hand, attempts are being made to exploit the formal nature of the model. One would like to prove theorems concerning the model which would allow one to derive properties of particular program-data pairs given their representation in the model. On the other hand, models of various programming languages are being constructed, and the problem of constructing processors based on the model is being investigated. One would like to be able to translate from actual programming languages into the model and vice versa. Given such processors, one could then manipulate the H-graph representation of a particular program according to the formal theory to produce semantically equivalent programs having particularly desirable properties.

Any model of the semantics of programming languages emphasizes in its structure certain semantic features at the expense of others. The model based on hierarchies of directed graphs presented here is no exception. Other models of semantics have tended to emphasize flow of control in programs and argument transmission to subprograms. Landin's lambda-calculus model of ALGOL,[1] for example, emphasizes argument transmission, restricts flow of control to function composition with recursion, and does not consider data structure. The directed-graph model of Narasimhan[3] emphasizes argument transmission and flow of control, representing the latter in a manner somewhat similar to that used here. The model of this paper emphasizes the structure of data and the hierarchical aspects of flow of control. It subordinates argument transmission by incorporating no built-in argument transmission method.

It is clear that the development and study of formal models of the syntax of programming languages has helped greatly to clarify the basic concepts in that area and also has led to the development of better methods for syntactic analysis in processors. It may reasonably be expected that similar benefits will be derived from the development of adequate formal models of the semantics of programming languages.

REFERENCES

1 P LANDIN
A correspondence between ALGOL 60 and Church's lambda-notation
Communications of the Association for Computing Machinery 1965 8 2 and 3

2 T B STEEL (editor)
Formal language description languages for computer programming
North-Holland 1966

3 R NARASIMHAN
Programming languages and computers: A unified metatheory
Advances in Computers Vol 8 F L Alt editor 1967

4 R W FLOYD
Assigning meanings to programs
American Mathematical Society Symposium in Applied Mathematics Vol 19 1967

5 J McCARTHY et al.
Lisp 1.5 programmers manual
Massachusetts Institute of Technology 1962

A flexible standard programming system for hybrid computation

by WOLFGANG GILOI, DIETER BECKERT and HANS C. LIEBIG

Technical University of Berlin
Berlin, Germany

Hardware structure of hybrid computer systems

The combined operation of analog and digital computers in hybrid computer systems requires a special hardware interface because of the different modes of operation and the different ways of data representation in both computers. Generally, in such a symbiosis all the functions of the analog computer, which are normally under manual control by the user, here have to be under control of the digital program, except for one case: once the digital program has started an analog computer run, the analog computer is on its own. In the case of combined simulation, i.e., if both computers operate simultaneously, from the viewpoint of the analog computer its digital partner now plays the role of a single, but very complex, computing unit to which some analog signals go and from which other signals come back.

From the viewpoint of the digital computer (and in the following we will always adopt this point of view), the analog computer and the interface are an entity representing an external process which has to be controlled. We assume that this process may include the various procedures listed in Table I, which all have to be managed by the digital program. (For notational convenience the word 'program' will always stand for the digital program, while we shall call the analog part of the entire program a 'setup.')

Table I

List of Procedures Which are Provided by the Hardware Interface Non-Time-Critical Functions of the Hardware Interface:

1. Control of the analog computer modes by the program
2. 'Run Time' selection (of one or more analog computer runs) by the program
3. Signalization of the actual start of an analog computer run (change of the integrator mode) to the program
4. Sensing of the current mode of the analog computer by the program
5. Signalization of the end of an analog computer run ('run time elapsed') to the program
6. Setting of potentiometers and subsequent checking by the program
7. Setting of analog switches by the program
8. Readout of analog outputs and potentiometer settings under control of the program and transfer of these data to the digital computer
9. Sensing of the state of Boolean variables occurring in the 'logic box' of the analog computer by the program

Time-Critical Functions of the Hardware Interface:

10. Multiplexing and sampling of analog data and conversion into digital form; transfer of these data into the digital computer at an arbitrary rate (serially)
11. Transfer of Digital data into the digital-to-analog converters (in parallel), conversion (and, in certain circumstances, smoothing) of these data
12. Transfer of overload messages and other error messages to the program
13. Transfer of external interrupts (arbitrarily programmable on the analog computer) to the program

All the procedures of items one through nine have to be executed while the analog computer is in the HOLD, RESET, or STANDBY mode. Hence, they are not very time critical. On the contrary, the procedures 10 through 13 take place while both computers are running, thus imposing on the program problems of real-time process management.

All this requires a digital computer that has process control computer capabilities such as a fast communication I/O-channel (preferably with direct memory access or a cycle-stealing mode, respectively), a hierarchic interrupt system, and control and sense lines. In addition to the real-time clock which is part of any analog computer, either the interface or the digital computer has to provide a second real-time clock (defining the 'frame time' yet to be explained).

In terms of harware only, the interface (plus analog computer) appears to the digital computer like one more peripheral device, yet much more complicated than the customary peripheral equipment. A much higher complexity of the data flow to and from that device is mandatory, varying between single control bits or sense bits and words or blocks of words to be transferred at a very high rate. In terms of the system programming, however, this particular peripheral device represents a complicated process, occurring in real-time and manageable only if a specially designed software interface is provided, which is as powerful as the hardware interface.

The requirements for a powerful software interface and its structure

The machine languages of digital computers which have the above-mentioned process control computer capabilities include instructions that energize control lines or check sense lines, handle interrupts, activate the I/O-channel and prepare its interlace hardware so that a block of data words can be put in or out, etc. However, it is much too tedious to program a hybrid computer that way (despite the fact that it has sometimes been done). In the case of the hybrid computer system about which this paper deals, a program written in machine code would require 35 instructions just for selecting an arbitrary analog computing element, and 25 more instructions to read the output of that element and transfer it into the digital computer. The setting of one potentiometer—including a subsequent check of the actual setting and the handling of possible error messages—takes about 150 instructions in machine code. Many of these assembly language instructions are not just simple, mnemonic words such as the arithmetical operation-code, but I/O-instructions which have to be specified, e.g., by a never-to-memorize six digit octal number. However, it is even worse if one leaves the subtle tasks of interrupt handling to the programmer.

All these problems may occur in process control applications too, but for such a special purpose, a program has to be written only once. Hybrid computer system programming, however, combines the short-term aspects of pure digital programming with the crucial difficulties of real-time data processing.

So, first of all, one has to eliminate coding by machine instructions. A comfortable software package for a hybrid computer system has to offer the possibility of writing any program in one of the customary problem-oriented languages that programmers are used to, for example, FORTRAN or ALGOL. Of course, such a language has to be augmented by a number of special subroutines. These subroutines take care of all the procedures listed in Table I. Furthermore, a special executive program is necessary which handles the synchronous operation of both computers and all the interrupts and error messages which may occur.

But that is not sufficient. Analog computer operators are used to having control of the entire system at any arbitrary time instant. They are able to interrupt the execution of a program in order to change potentiometer settings as well as the whole setup and restart or rerun thereafter the program. A hybrid computer system should provide the same possibilities, but this cannot be done while the system is under control of the digital program. Conversely, the program may sometimes need the help of the human operator (at least as long as the automatic patching problem has not been solved yet). The first demand to hand over the control of the system to a human operator or to receive it back at any arbitrary moment is unique. If this is granted, there are actually three parties involved alternating in the control of the system: the digital computer, the analog computer, and the human operator. This gives us additional reason for using the term 'software interface' as a supplement to the 'hardware interface,' meaning that only the proper design of both can make a hybrid computer system manageable. Additionally, some special utility programs are very helpful which perform an automatic checkout of the analog setup and of the entire system, thus detecting, indicating, and diagnosing errors and component failures.

Hence, a hybrid computer system 'housekeeping' software package should at least consist of the following programs (names have been assigned to the various programs which we can refer to):

1.	HARTRAN	(*h*ybrid-procedure *a*ugmented *r*eal-*t*ime FORT*RAN*)
2.	HYTROL	(*hy*brid system con*trol*)
3.	ACID	(*a*nalog *c*omputer and *i*nterface *d*iagnosis)
4.	STATEST	(*sta*tic *test* of the analog setup)
5.	HYBRID LIBRARY PROGRAMS	

HARTRAN denotes the set of all hybrid subroutines by which the problem-oriented language (in our case a special FORTRAN version) has to be augmented. If the software interface is organized the way it will be subsequently described, HARTRAN includes also the executive program. HYTROL provides the required man-machine-interaction. ACID is a necessary augmentation of the debugging programs which are already existing for the digital computer. STATEST enables the digital computer to check-out the analog setup and to detect and indicate set-up errors as well as component inaccuracies or failures. The HYBRID PROGRAM LIBRARY encompasses a number of hybrid standard programs which usually cannot be found in the standard program library supplied with the digital computer.

It is common practice to compose a complex programming system of several subsystems, because it gives a much better way to implement, improve, maintain, and change such a system. Doing so, one has to take care, of course, of all possible interconnections between the various subsystems. In order to facilitate the realization and the use of the subsystems, we define first of all a certain set of modules which are common components in all of them. The modules occur in three different forms:

PROGRAMMED OPERATORS (POPs)
STANDARD HARTRAN SUBROUTINES (SUBs)
SPECIAL HARTRAN FUNCTIONS (FUNCs).

A POP is a subroutine that has a mnemonic name, like any other macroinstruction, combined with a parameter by which the execution is defined. The SUBs are standard subroutines of the HARTRAN compiler, i.e., they don't have to be declared and can be called by a CALL statement together with their name and an arbitrary array of parameters. The FUNCs designate procedures by which the output of an analog component or a Boolean variable is fetched from the analog computer. The actual variable receives the name of that function, and under this identifier it can be a member of any arbitrary arithmetic or Boolean expression (like the standard functions in the usual FORTRAN language).

SUBs and FUNCs, for example, take care of all the procedures listed in Table I and others. POPs are first of all used within HYTROL and STATEST, and the ACID complex of test programs rely on SUBs as well as on FUNCs. The special library programs are written in HARTRAN. We will talk about these modules in more detail when we come to the description of the various subsystems.

What problem-oriented programming language should be used? The standard FORTRAN language (especially FORTRAN II) has some shortcomings. For example, it does not provide the possibilities of recursive call of subroutines, instructions for bit manipulation, and handling of interrupts. Notwithstanding, for two reasons, we selected FORTRAN. The first and most important reason was that FORTRAN is the most often used programming language. The intention of this paper is, however, not to describe just one of many possible implementations of a hybrid system software interface, but to propose a standard type of software interface which can be implemented on any system. Therefore, it has to be independent of particular assembler languages and it has to be based on the problem-oriented language most in use.

The second reason was an individual one. For our computer (SDS 930), a special FORTRAN version exists, called REAL-TIME FORTRAN, which does include all the above mentioned features, especially the possibility of connecting subroutines which respond to interrupts. Of course, this is no comfort to somebody who has to rely on the standard FORTRAN II version, but, even then, the construction principles outlined in this paper can be applied. HARTRAN would in this case stand for '*h*ybrid-procedure *a*ugmented FO*RTRAN*,' and this programming system may lack some valuable conveniences, but it will still be feasible.

The organization of the software interface; HARTRAN and HYTROL

The two possible states of a hybrid system

From the discussion in the above section, the concept of organization of a hybrid computer software interface becomes quite obvious: The system as an entity (hardware and software) has two possible states, namely:

(I) the CONTROL state
(II) the RUN state.

In the CONTROL state a number of standard subroutines can be called within the framework of HARTRAN. The subroutines execute the special hybrid procedures which we have listed in Table I under items one through nine, and some more. Additionally, by calling a special subroutine by the name HYTROL, control is transferred to the console typewriter or any other arbitrary input device, enabling the human operator to interrogate or control the system. HYTROL, on the other hand, has to provide instructions by which a return to the HARTRAN program is possible.

Another most important HARTRAN subroutine called OPERUN switches the system from the CONTROL state into the RUN state. By the CALL OPERUN statement two different processes are started simultaneously:

a. An analog computer run is initiated (switching the analog computer from the STANDBY, INITIAL CONDITION or HOLD mode, to the *OPE*RATE mode).
b. In the digital computer, a procedure called *RUN* is started that initiates a continuous scanning, sampling and converting of analog data, a manipulation of those data, and transfer of the digital results into the digital-to-analog converters.

At this point we have to comment about the two different classes of hybrid computation and their consequences on the organization of the data transfer between both computers. In the first class of operation, both computers operate alternatively, while in the second class, both computers execute a hybrid program simultaneously.

For operations of the first class, the programmer should be given the flexibility of a random input or output of analog data; i.e., each I/O procedure has to be programmed individually, and every time the multiplexer or DAC addresses can be arbitrarily chosen.

Unlike the first case, the second case may be extremely time-critical, especially if a combined simulation of a system with high natural frequencies has to be performed in real-time. Since in this class of operation an input or output sequence of analog data may occur hundreds or thousands of times during one computation run, it would be unnecessarily awkward and time consuming to program this by a random access which needs to declare for every input or output the multiplexer or DAC addresses. Therefore, a procedure should be available which performs automatically and periodically a certain I/O-sequence. The first address and the length of the input and output sequences have to be declared just once (in a program header). All the time during the execution of that program, a fixed sequence of multiplexer input lines are scanned and a fixed sequence of DACs are loaded periodically. If the channel has the capability to transfer data blocks autonomously to and from the memory in a cycle-stealing mode bypassing the CPU (we call this the interlace feature), this particular I/O-mode provides not only a most efficient way of programming, but also the fastest possible way of execution.

HARTRAN subroutines are available for both ways of inputting and outputting analog data. It has to be emphasized that this way of programming can be used for any computer, while the actual subroutines, of course, depend on its specific structure. In the following, we shall find some more SUBs which are a result of the specific hardware structure of our hybrid computer system. But this is not contradictory to our claim of proposing in this paper a standard software interface. The only thing we have to try is to assume a hardware structure as general and flexible as could be. The SUBs and the corresponding POPs which may then be required for a less flexible system are a subset of what we shall define in the following. The particular subroutines which can be called as SUBs or POPs have to be written for the individual system anyway. Following this philosophy, for example, it does not matter whether or not the multiplexer inputs are equipped with parallel track-and-hold circuits, or whether the DACs have two buffer registers or only one. Of course, the corresponding subroutines have to be written differently, but the structure of the software interface and the programming language are not affected. The same holds for some unique hardware features of our system.

The control state and HARTRAN

In the following we shall list all the standard procedures which can be executed by a HARTRAN program. On the left side we list the respective format, and on the right side we give a short specification of the procedure. Furthermore, the SUBs are classified by their function and a comment is made on each class.

Analog Computer (AC) Mode Control

CALL CON	::	the next run will be 'continuous operation' unlimited in time
CALL CON H	::	the next run will be 'continuous operation' until the selected time has elapsed; after that the AC goes in 'hold' and continues when a new 'operate' instruction is received
CALL REP	::	the next run will be 'repetitive operation'
CALL REPH	::	the next run will be 'repetitive operation'; after the first run, the AC goes in 'hold' and starts the next repetition run only on receipt of a new 'operate' instruction
CALL ITR	::	the next run will be 'iterative operation'

CALL ITRH :: see REPH and replace 'repetitive' by 'iterative'

CALL STY :: if the AC was in 'hold' it is now switched to 'reset' or 'standby' (both are synonymous)

CALL SRTN ('rt', 'ot', 'ht') :: setting of the 'normal' run timer

CALL SRTC ('rt', 'ot', 'ht') :: setting of the 'complementary' run timer, 'rt' is reset time; 'ot' is operate time; 'ht' is hold time (all declared in milliseconds)

Comment: A model of the AC mode control is a (virtual) manual control that has six push-buttons and six thumbweels. The push-buttons are labeled: continuous operation (CON), repetitive operation (REP), iterative operation (ITR), operate (OP), hold (H), and reset or standby (STY). By pushing one of the first three buttons the mode of the next computer run is prepared, and by pushing the OP button this run is actually started. If, in addition to REP or ITR, the button H is switched on, we have the 'single run' mode. The combination of CON and H interrupts the continuous operation after the selected run time has elapsed; the AC goes in the HOLD position from where it can be restarted in order to continue the operation. By the (virtual) thumbweels, the (normal) 'run timer' can be set, defining the three phases of a repetition cycle individually. The 'complementary run timer' setting is only required for the iterative operation. Though this mode does not make much sense in a hybrid computer system, it is part of most analog computers and shall thus be taken into account.

Setting of Potentiometers and Function Generators

CALL POTSET ('addr.u', 'value u', 'addr.v', 'value v', . . .) :: setting of an (unlimited) number of potentiometers

CALL POTSETL ('addr.u', 'value u', 'addr.v', 'value v', . . .) :: setting of potentiometers with a printout of the actual addresses and settings

CALL FGSET ('value 1', 'value 2', . . .) :: setting of function generators; 'value 1', 'value 2', . . . are the (fixed) breakpoint values of the function given as a decimal fraction of the reference (e.g.: + .4875).

Comment: 'addr.' may be any combination of letters and figures, depending on the AC address system; 'value' is a 4-digit decimal number following the decimal point (e.g.: P 127, .0178). For the error messages see a later section (HYTROL). FGSET, of course, only makes sense if the AC has digitally settable function generators. (In our system, we developed these units ourselves.)

Control Lines

CALL SET ('x', 'y', 'z', . . .) :: The control lines listed as parameters of the expression are set to TRUE.

CALL CLR ('x', 'y', 'z', . . .) :: The control lineslisted as parameters of the expression are set to FALSE (CLR = 'clear').

Comment: The control lines are used to set or reset flipflops in the AC logic box.

Selection of Analog Components and Senselines; Functions

POT ('addr.') :: The potentiometer with a named address is selected, and its output is a variable of the program identified by POT ('addr.')

AMP ('addr.') :: The amplifier (multiplier, function generator, etc.) with the named address is selected, and is a variable of the program identified by AMP ('addr.').

SL ('x', 'y', . . .) :: The logical values of the sense lines 'x', 'y', . . ., are composed by 'and' and the result becomes the logical value of the function.

Comment: As mentioned above, functions can be variables of arithmetical or Boolean expressions. One can write, for example, a FORTRAN IF statement as follows (x being another variable):

```
IF (AMP (47) — .5800 * X) 10, 10, 20
```

If 'x', 'y', 'z', are addresses of sense lines, one can write, for example, (without having to declare 'x', 'y', and 'z' as LOGICAL)

```
If (SL(1) . OR . . NOT . SL(2, 3) 1, 1, 2
```

where the expression of the IF statement is
$\overline{\text{SL1} \vee \text{SL2} \wedge \text{SL3}}$!

Entry to HYTROL

CALL HYTROL	::	The control and test program HYTROL is called and the system control is handed over to the human operator

Frame Time Selection

CALL FTS('time')	::	In preparing the run state, the time of a digital computation frame (see next section) is selected by setting a special real-time clock ('time' = 3 place decimal number giving the time in milliseconds).

DAC Mode Selection

CALL MODC	::	Simple conversion (the AC reference is also the reference for the digital-to-analog converters (DACs).
CALL MODM	::	The DAC reference voltage can be an arbitrary analog variable, hence, conversion is combined with an (analog) multiplication.
CALL MODI	::	The DACs combine the conversion with a straight-line segment interpolation (or extrapolation).

Comment: It is another special hardware feature of our system that the digital-to-analog converters (DACs) can operate in three different modes (we designed them ourselves for that particular purpose[1]). The first mode is just simple conversion. In the second mode the converters may be used as 'multiplying DACs' providing the multiplication of the converted digital output with any arbitrary analog variable. A considerable number of analog multipliers may be saved, and the errors may be reduced (since the MDACs are more accurate than analog multipliers).

The output time function of a usual DAC is a 'staircase' function, i.e., the output voltage is constant as long as the digital input stored in the DAC register is not replaced by a new one. In the third mode, our interpolating (extrapolating) DACs replace the staircase function by a straight-line segment interpolation (or extrapolation) between the data points, resulting in a better (and smoother) approximation of a continuous signal of which the data points are samples. For an interpolation, the program has to calculate the increments between a current output and its successor. If the successor is not available to the program, the interpolation must be replaced by an extrapolation based on the current output and its predecessor. A variation of the time interval between two data points (= frame time) is automatically taken into account. If the digital-to-analog converters have only the DAC and MDAC modes, the MODI subroutine is immaterial; if there is only the normal DAC mode, all the SUBs of this section are immaterial. The mode specification holds for all DACs. If only part of them shall multiply, the others have to be patched to the reference. If only part of them shall interpolate, the mode is MODI, and the ones which have not to interpolate receive simply the increment zero. In our case, this is particularly simple, as we have a 24-bit word format where the first 16 bits are the value, and the last eight bits are the increment.

Asynchronous Data Transfer

CALL AD ('x', 'addr.x', 'y', 'addr.y', . . .)	::	transfer of data blocks from the AC to the DC. 'x', 'y', . . . are variables or expressions of the HARTRAN program identified by the numbers of the multiplexer input lines by which the variables are fetched.
CALL DAC ('x', 'dest.x', 'y', 'dest.y', . . .)	::	transfer of data blocks from the DC to the AC. 'x', 'y', . . . are variables or expressions of the HARTRAN program; 'dest.x', 'dest.y', . . . are the destination addresses.
CALL DAI ('x', 'INCx', 'dest.x', . . .)	::	transfer of data blocks from the DC to the AC. 'INCx', 'INCy', . . . are the increments of the variables 'x', 'y', . . . between two data points; for the other parameters see CALL DAC.

Comment: DAC has to be used in connection with MODC or MODM; DAI is only used in connection with MODI.

Switch from the Control State to the Run State

CALL OPERUN :: The system is switched into the run state, either for starting an analog computer run only (with no simultaneous digital operations) or for a combined operation of both computers.

Comment: Most analog computers have the possibility to change all the integrator time constants simultaneously by a factor 10 (e.g., by a '10 times faster' or a '10 times slower' switch). If this should be done under control of the program, the OPE operator (respectively the OPERUN SUB) may include a parameter which specifies the respective integration rate. This has the advantage, that nothing else in the program has to be changed. We suggest this as a possibility but we don't feel that it makes much sense to have this parameter under control of the program.

The run state

Figure 1 illustrates the general scheme of the RUN procedure executed by the OPERUN subroutine. Right after the start of the procedure by a CALL OPERUN statement, the subroutine sends (after having executed some preliminary instructions) an OPE instruction to the AC, hence starting an analog computer run. When the AC integrators really start, the AC sends a "start" interrupt to the DC causing the begginning of the first computation frame. By these means we obtain well-defined timing of computers. Simultaneously, both real-time clocks (the 'run time' and the 'frame time' clock) are started.

Every time the selected frame time has elapsed, the 'frame time' clock sends an 'end ot frame' interrupt (EFI) to the DC, indicating the end of a frame and the beginning of the next one. Each EFI starts a combined input/output of analog data blocks as illustrated in Figure 2. Length and formation of the respective

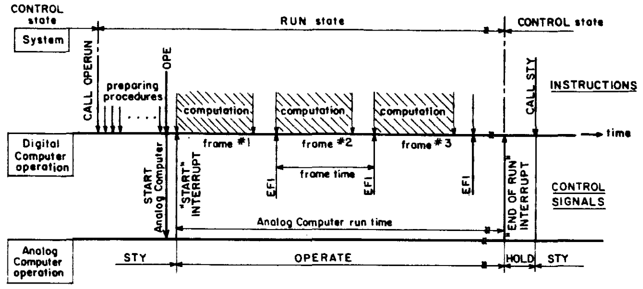


Figure 1—General scheme of the RUN proceduce

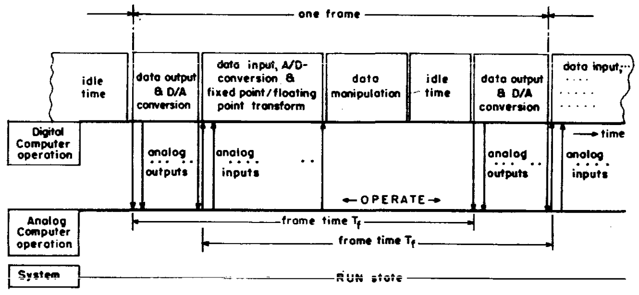

one frame
idle time
data output & D/A conversion
data input, A/D-conversion & fixed point/floating point transform
data manipulation
idle time
data output & D/A conversion
data input,...
Digital Computer operation
analog outputs
analog inputs
analog outputs
analog inputs
time
OPERATE
Analog Computer operation
frame time T_f
frame time T_f
System
RUN state

Figure 2—Organization of a computation frame

block have been specified together with the various source and object addresses by calling the DADC or DADI subroutines to be explained in the following.

Eventually, the selected run time of the analog computer will have elapsed. The AC goes at that moment in the HOLD mode and sends an 'end of run' interrupt to the DC causing the program to switch back to the control state. The 'start' and the 'end of frame' interrupt are of the same priority (higher than the 'end of run' interrupt).

Some additional comments should be given on the organization of a computation frame. Each frame starts with an output procedure, whereas the outputs are the results of the computations in the preceding frame (in the case of the first frame they are initial values). This happens as fast as the digital computer can put out these data (in our system it takes two memory cycles = 3.5μs for each DAC). The DAC settling time (which in our case[2] is about 0.5μs) is not relevant in this context, as in fact the DACs could be loaded simultaneously if the DC could do that.

Right after the output sequence follows the input sequence (for the current frame). Depending on the throughput rate of the multiplexer/converter the ADC may feed-in data at a lower rate than the DC could handle. In such a case, one can use the idle time between two inputs for performing the necessary fixed-point/floating-point transformation on the current input. While we started with this approach, we have now an ultra-fast multiplexer converter in our system that has a higher maximum throughput rate than the DC can handle. (This multiplexer/converter is at present the fastest commercially available device of its kind and was developed by ourselves.[2] It has a 15-bit accuracy at a maximum throughput rate of 500,000 per second.) Therewith, we put in data at the maximum possible speed of the DC channel and perform the fixed-point/floating-point transformation later on. Though the total execution time is not reduced, the skewing errors become negligible, though parallel track-and-hold circuits (which would, in turn, introduce some other inaccuracies) are not used.

After the output/input sequence and the associated fixed-point/floating-point transformations have been finished, the execution of the computation frame begins. Depending on the chosen frame time, a time interval follows during which the DC is idle until the EFI occurs. It is at least very difficult, if not impossible, to estimate the exact execution time a priori in order to choose the frame time so that any considerable idle time is avoided. If the computation frame, for example, includes branching instructions or external interrupt, the execution time of a frame may not even be constant. This dilemma is solved by a special procedure within the RUN subroutine which measures the idle time, (the DC is not actually idle, but counting clock impulses). The minimum idle time of a computer run is stored and may be printed on request. Thus, after one test run the programmer can determine the right frame time. If a negative idle time occurs (i.e., if the frame time chosen was too short) an error message is printed in any case.

The part of the program which follows the OPERUN statement and which constitues what we call the 'computation frame' may include any regular FORTRAN expression as well as some particular SUBS. SUBs which can be called within this program are listed as follows:

Synchronous Data Transfer to and from the AC

CALL DADC ('DA', 'NDA', 'AD', 'NAD') :: data transfer in both directions synchronized by the EFI according to Figure 2.
DA : first memory address of the output data block
NDA : length of the output data block
AD : first memory address of the input data block
NAD : length of the input data block
The DACs have to be preset either to MODC or MODM.

CALL DADI ('DA', 'INC.', 'AD', 'NAD') :: like CALL DADC but with the DACs in MODI.
'INC' : first memory address of the block of increments

Comment: CALL DADC and CALL DADI can only be called as part of the RUN subroutine. NAD = 0: only transfer D-to-A; NDA = 0: only transfer A-to-D. Error messages are given if illegal addresses or numbers of data words are specified.

External Interrupts, Sense Lines and Control Lines

CONNECT SUB 'no' (FINT('no') . . .) :: SUB 'no' is the name of a subroutine written by the programmer which is sensitive to 'free' external interrupts (FINT), i.e., the subroutine is executed when the interrupt occurs. Notice that SUB 'no' is not a standard HARTRAN subroutine.

CALL SET ( ),
CALL CLEAR ( ), SL ( ) : See previous section

Comment: The sense line and control line SUBs are used as in the control state. To write particular subroutines which are sensitive to external interrupts and to connect them to the program by a simple CALL or CONNECT statement is only feasible in special FORTRAN versions.

Return Into the Control State

CALL HOLD :: The analog computer is switched from the OPERATE to the HOLD mode and the system goes into the control state.

CALL HYTROL :: HYTROL can also be entered from the run state, switching the system necessarily to the control state.

Comment: If an overload or a channel error (erroneous multiplexer or DAC addresses or ADC overload) occurs, an interrupt with highest priority is sent to the DC, causing the system to return into the control state. Either OVERLOAD or CHANNEL ERROR is printed, and after that HYTROL is automatically called giving the user an opportunity to check for the error reasons or to restart the program execution.

Software error compensation

Intrinsic error sources of a combined computer system are:

(i) the necessary sampling of analog values transferred to the DC and a subsequent imperfect smoothing of the digital results which go to the AC (the DACs approximate the theoretically required ideal low-pass filter very poorly)

(ii) time delay between corresponding input and output vectors because of the execution time of the digital computation frame

(iii) time shift between the various components of input and output vectors because of the serial input/output.

We call the respective errors caused by these three sources the 'sampling errors,' the 'delay errors' and the 'skewing errors.' The skewing errors can only be avoided by appropriate hardware measures such as parallel track-and-hold circuits at the multiplexer inputs and double-register DACs, or—as in our case—by an I/O transfer rate that is so high that no appreciable errors occur. Some authors have suggested reducing sampling and delay errors by utilizing additional hardware in the analog computer, however, these methods are tedious and too expensive to be of practical use. Giloi has shown in an earlier paper[3] that a much better error compensation can be obtained by a digital filtering algorithm which is part of the computation frame. This method reduces sampling and delay errors simultaneously by an arbitrary high degree and is very easy to implement.

It has been shown in the quoted paper[3] that the error compensation is almost as good when using nonrecursive filter functions as in the case of recursive functions, yet nonrecursive functions do not cause stability problems. Therefore, HARTRAN includes a 'filter' algorithm of the kind

$$G(z) = a_0 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3} + a_4z^{-4} + a_5z^{-5} + a_6z^{-6}$$

which may be executed in the context of the RUN subroutine by the statement

CALL ECF
('a_0', 'a_1', 'a_2', 'a_3', 'a_4', 'a_5', 'a_6') :: digital error compensation filter passed by all the output-bound data

The maximum order of the filter is six, but by setting some of the coefficients to zero any lower order may be obtained. One important parameter of the filter does not have to be declared, as it is the 'sampling interval' which is equal to the frame time and, hence, is known to the filter algorithm by the CALL FRA statement. If the frame time is changed, the filter subroutine takes this automatically into account. (For the values of the coefficients $a_0, a_1, \ldots, a_6$ see Giloi's paper.)[3]

The control state and HYTROL

By calling HYTROL the system is put into the control state (no matter in which state it was) and the system control can be performed by the human operator in a conversational mode. The program begins the dialogue by typing HYBRID CONTROL and in the next line a C. That means that the user is asked to specify the control medium, i.e., the device by which he wants to communicate with the system. If he types TY1, he continues using the console typewriter. In the case of a spacious installation or, even more, if the digital and the analog part of the system are located in different rooms (as in our case), it is very convenient, if not inevitable, to have next to the analog computer console a second typewriter which is selected by typing TY2. Card readers (CR) and paper tape readers (PR) may also be used, but only in the case when the system returns eventually to the HARTRAN program (i.e., the last statement has to be either RET or BEG). If an L is added to CR or PR, the instructions are printed. Any other statement than one of the set {TY1, TY2, CR, CRL, PR, PRL} typed after the C causes a SYNTAX ERROR message as well as any other violation of the HYTROL syntax.

After that, the user can start any controllable function of the system. For this purpose he has a set of MACROs at hand.

For the analog computer mode selection we have the following set of MACROs

{CON, REP, ITR, CONH, REPH, ITRH, STY}

corresponding with the SUBs of an earlier section. An analog computer run in the selected mode is actually started by typing OPE. This is in correspondence to what the OPERUN subroutine is doing.

For the setting of a potentiometer, one has to type

POT 'name' 'value'

(blanks are arbitrary). The reader may notice that the potentiometer address is now part of the name, as the POPs allow only to specify one parameter (the value, defined as in the POTSET SUB).

The same feature holds when the AC readout system is activated by a MACRO. Here we write

SEL 'name'

whereas, again, 'name' is the analog component which has to be selected (for example P 137, A 72). The output of the selected unit is converted and printed as a 4-place decimal number plus sign (e.g., + .dddd).

Sense lines and control lines can be checked (CHK) or activated (SET/CLR), respectively, by the following set of MACROs

{CHK 'name', SET 'name', CLR 'name'}

For the setting of the 'normal' and the 'complementary' run timers and the 'frame time,' we have the MACROs RTN'ddd',

OTN'ddd', HTN'ddd'	::	(reset time normal, operate time normal and hold time normal)
RTC'ddd', OTC'ddd', HTC'ddd'	::	(reset time, operate time and hold time of the complementary phase)
FTS'ddd'	::	(frame time selection)

'ddd' stands in all cases for a decimal number which specifies the time in milliseconds.

Finally, we have two MACROs which provide a return into the HARTRAN program (which we may have left by a CALL HYTROL statement). When typing

BEG :: begin

the HARTRAN program is started anew at the beginning, while by typing

RET :: return

the HARTRAN program continues by executing the SUB which follows the CALL HYTROL statement (the point at which the program was left).

The execution of a procedure is actually started by the subsequent carriage return. If a comma is typed, the procedure represented by the MACRO in the preceding line is once more executed. HYTROL provides some error messages and some diagnoses. Possible error messages are

SYSTEM NOT READY	::	if the analog computer and/or the linkage is not switched on
SYNTAX ERROR	::	in case of any violation of the HYTROL syntax
OVERLOAD	::	if the AC has only a common but no individual overload indication

Diagnosing error messages are

OVERLOAD 'name'	::	if the AC has an individual overload indication, 'name' means the name of the overloaded component
ADDRESS ERROR	::	this message follows a POT, SEL, CHK, SET, or CRL if the name in this statement is none of the legal names of all the analog components, sense lines, and control lines.
SELECTION IMPOSSIBLE	::	in this case the name in the preceding MACRO was legal but the associated component is not existing in this particular installation.
SERVO SETTING IMPOSSIBLE	::	follows a POT operator if the servo does not work
TRIED THREE TIMES, DIFFERENCE ± .dddd	::	the servo works, but after three trials the difference between the nominal value and the actual setting is still more than 0.04 percent (this threshold is arbitrary). The maximum deviation is printed.

The reader may notice that by the above listed MACROs, any parameter of a HARTRAN program can be arbitrarily changed, and thus, all control and possibilities of on-line debugging are provided. If the user should decide to terminate the hybrid computation (because he found a mistake in the program or a component failure which cannot be repaired immediately, or for any other reason) he types.

M

Hence, the monitor system of the digital computer is called. The DC may start to execute any arbitrary program (or a batch of programs) under control of the monitor. This program batch may have nothing to do with the hybrid program, or it may contain standard I/O programs which are used in order to process the results of the hybrid computation (e.g., printer, plotter, or display routines). If the HARTRAN program ends with the last two statements

```
CALL HYTROL
M
```

the system returns automatically into the monitor mode.

Time-sharing of the digital computer

With respect to digital computer utilization, hybrid computation is extremely inefficient and, thus, expensive. The setting of a servo potentiometer for example requires some seconds, the setting of some tens of potentiometers adds up to minutes, during which the DC is idle. If the user is in the process of debugging his hybrid program on-line, it is even worse. In the case of an undebugged digital program, on the occurrence of an error, this program is dumped out, and the programmer may think about his mistakes without locking the computer. The hybrid programmer, however, will start to use HYTROL (since he has this wonderful tool available) in order to find out what is wrong. During the many seconds or minutes of meditation, he will hardly release the system. The worst of all cases occurs when he must halt to change his analog setup (as analog computer users are very likely to do).

If the monitor is loaded together with a 'background' batch of programs, the user can at least immediately restart the execution of the background batch (by the M operator) when he realizes that for some reason he cannot continue with his hybrid computation. But there are two problems: Once he has lost the access to the digital computer it will become difficult for him to get it back. The second problem is that in a so-called balanced system, the DC has hardly core memory enough to accommodate the resident part of the monitor, at least one background program, possibly a compiler, and in addition the HARTRAN program and HYTROL. (OPERUN and HYTROL alone take about 2.7 K of core memory.)

If the installation includes a rapid access disc or drum, the second problem could be solved by swapping the hybrid program for the backgorund process and vice versa. From there it is only a minor step to time-share the digital computer all the time the hybrid program is in the HYTROL mode. Since there are only two processes competing for CPU time, a relatively large time slice can be allotted to the background program without causing discomfort to the user of the hybrid system.

The simplest way to implement such a system is to modify HYTROL so that when HYTROL is called it sends the C message and calls the monitor right after that to start executing the background batch. After a certain time (e.g., one second) given by a real-time clock (e.g., that of the AC run-timer), the background process is interrupted and HYTROL asks whether the hybrid system operator has in the meantime completed a HYTROL statement. If the answer is yes, this statement is executed and the system returns to the background process, as it does directly if there was no HYTROL statement completed. If the statement is RET or BEG, the background process is swapped out and the entire hybrid program is swapped in from the disc. Therefore, the digital computer is only exclusively assigned to the hybrid process when the latter is in its real production phase. Notice that for the implementation of such a system only a small part of the entire program package—namely HYTROL—has to be modified, one of the advantages of the modular structure of our software interface.

STATEST

STATEST is a special program which executes automatically a static check of the analog setup, hence detecting patching errors as well as failures or poor performance of components. By virtue of a built-in optimization strategy, the best suited test values are automatically evaluated by STATEST. Everybody who is familiar with the static check procedure on an analog computer knows that this is a most crucial point. In the case of STATEST, however, all that has to be done is to give the program a description of the analog setup (the analog part of the hybrid program in treatment). As far as we know, these features are unique.

The setup description consists of a list of connection statements, one for each component, which are written in the form of pseudo-equations. On the left side of such an equation the name of the 'object' is listed the output of which has to be checked. On the right side, the names of all source elements are listed which contribute to the input of that particular unit, combined by the arithmetic expression which this unit is performing. The names of the various units correspond with the addresses of the analog readout system. Variables (represented by the name of theii source) which are fed to inputs with gain factors different from one have to be multiplied in the equation by the respective gain.

In the case of a multiplier, for example, which may have the name M19 and which may multiply the outputs of summer S46 and of function generator F3, we write

$$M19 = S46 * F3$$

(spanks are arbitrary). If the function generator input comes from integrator 17, we write

$$F3 = F3\,(17)$$

Hence, F3 denotes both the name of a component (on the left side of an expression) and the corresponding function (on the right side of an expression) which has to be specified by a table and which has the source element as argument. Potentiometers and digital-to-analog converters (DACs) are not source elements, but are listed as multiplicative coefficients of the associated variables, represented by their name and not by the value to which they are set. Thus the setting of potentiometers and DACs may be arbitrarily changed without any change in the connection statement list. The current settings can be found in the POTSET list which has to be read-in or typed-in anyway. Figure 3a gives an example for one particular branch of the setup. In this case, we have to write

$$I17 = 10 * I16' + P35 * NR + 10 * I17' + F3 + S23$$

Notice that the notation may be recursive, if there is a feedback from the output of the object to its input, except in the case of integrators (as in our example). In the static check mode, all integrators are changed into summers, the output of which yields the (negative) sum of all inputs, while the proper integrator output is substituted by a test value taken from the reference directly or via a potentiometer, or even from other components such as, for example, DACs. This is indicated in Figure 3a, and in this case we have to extend the connection statement list by the following two statements

$$I16' = P1 * NR \text{ and } I17' = P2 * PR$$

etc. (PR = positive reference, NR = negative reference).

When STATEST is activated (after the complete setup description, including the POTSET and FUNCTIONS list, was put in), it switches the AC mode to 'static check' and starts setting the potentiometers to

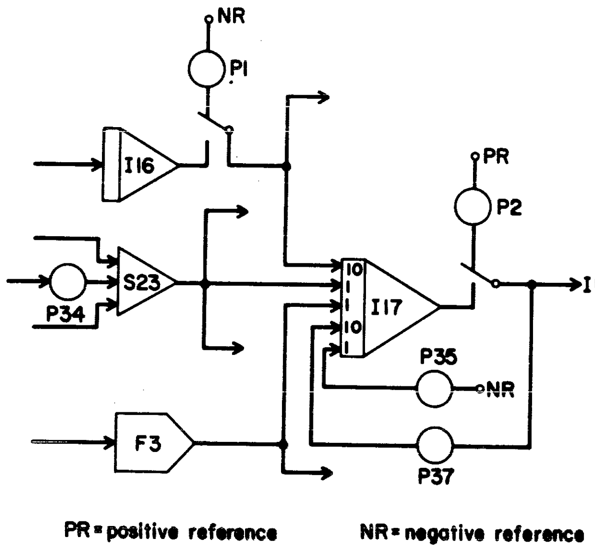


Figure 3(a)—Example of a particular setup branch represented by one connection statement

the values of the POTSET list. After that, it focuses its attention on the first pseudo-equation of the list. As a first step, it measures the output v_0 of the object of this equation and checks whether it is in the range

$$.05\ v_{ref} \leq v_0 \leq v_{ref}\ (v_{ref} = \text{reference voltage}).\ (*)$$

This may not be the case, and thus, STATEST has to take steps in order to enforce the validity of this condition. For this purpose, a search procedure is started which goes through all the branches of the tree which is a graphical representation of the input/output relations

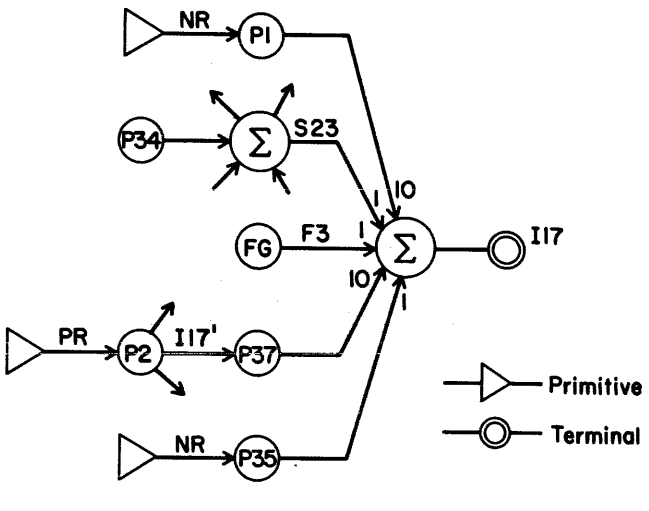


Figure 3(b)—Logical representation of the connection statement given in the text (Associated with the setup of Figure 3(a) of the particular unit under consideration. (For the given example, such a tree is shown in Figure 3b. The nodes of the graph are denoted by the algebraic operations and the branches by the name of the variables.) The program checks all the potentiometers in the respective equation (the potentiometers that specify the test values included) to see which of them has maximum influence on the terminal node when being changed. Thereafter, the setting of the potentiometer for which the object output is most sensitive is changed until condition (*) holds.

As STATEST has to traverse through several branches in order to find such a potentiometer, the operations of the intermediate nodes indicate the direction of the change (e.g., in the case of a divider a coefficient of the nominator would have to be increased in order to increase the output of the unit while a coefficient of the denominator would have to be decreased, and vice versa).

In any branch of the tree, the search procedure ends at the first potentiometer or a 'primitive' like the reference. In any branch, up to three components can be taken into account (this is a matter of the available core memory). STATEST does not care if there are overloads in parts of the setup other than the particular one which is currently under consideration. Hence, the convergence of the procedure is always guaranteed.

Once condition (*) holds for the current object the outputs of all the source elements are measured too, and the nominal results of the algebraic expression represented by the connection statement is calculated and compared with the actual (measured) result. Any deviation exceeding a given boundary ϵ^* results in an error message together with a printout of the error magnitude. The parameter ϵ^* is individually evaluated by the program for each one of the respective objects as a multiple of a common error level ϵ. Hence, the number and gain factors of inputs of the object are taken into consideration (e.g., a gain of 10 leads to an ϵ^* that is 10 times greater than ϵ, etc.). The basic 'precision parameter' ϵ depends, of course, on the particular analog computer and has to be declared by the user.

Eventually, when the program has gone through all the connection statements, all potentiometers are reset according to the POTSET list. Since that procedure is double-checked by the POTSET SUB, it is made sure that no new errors are introduced after the STATEST procedure has been finished. There are some more important details of this very sophisticated program which cannot all be mentioned in this paper. Our current STATEST version takes 10K of core memory for the interpreter plus some additional memory space for the connection statement list (approximately 1K for a typical 100-amplifier-program). But it could easily be shortened to run on an 8K configuration. STATEST includes also a routine which searches for hidden algebraic loops and gives a message if there is one. Furthermore, it can be easily connected with a program that calculates the setup (such as APACHE[10]). Of course, STATEST is only loaded into core when needed.

ACID and the PROGRAM LIBRARY

Most of the interface and analog computer functions could be checked by using HYTROL, but not all of them. Furthermore, if one wants to check out systematically the entire system and its reliability, it would be much too tedious to do it that way. All these tasks have to be automatically accomplished by special test programs. Naturally, the number and kind of such programs depend on the specific structure of the analog computer and the hardware interface, so that it is not possible to suggest a general concept as in the case of the operating system. Notwithstanding, we will give in the following a list of the ACID programs available and a short description of their function in order to indicate what these programs are about. It is almost needless to say that these programs use the same modules (SUBs, POPs, and FUNCs) as all the other parts of the software package.

SELEC	::	Test of the AC selection system (ACSS). Selects for a designated number of times every AC component. Each time the actual contents of the ACSS address register is checked via a sense line feedback. In case of a deviation, an error message is printed together with the actual address and number of iteration. It is also printed if the selection of a component is impossible e.g., if this component does not exist.
MODE	::	Test of the AC mode control. All possible AC modes are activated, and the actual mode is checked by sense lines.
CLOCK	::	Test of the AC run timer and the frame time clock. Both timers are set so that the ratio of the run time with respect to the frame time is an integer greater than one. The number of end-of-frame interrupts (EFIs) which fall between a 'start' and a 'end of run' interrupt is counted. If this number does not correspond with the time ratio, an error diagnosis is given. The procedure is executed for various run time and frame time clock settings.

READ :: Test of the AC readout system. Test values are given via a DAC into the AC readout system and read back via the ADC into the DC (the AC readout system has a special dummy address for this purpose). Deviations which exceed a designated level are printed together with the DAC and multiplexer address used.

POT :: Repeated test of potentiometer settings. Arbitrary parameters are the number of repetitions, the increments of the pot settings and an error threshold. The program sets every pot to a certain sequence of values starting with zero and increasing by the chosen increment (until .9999). This procedure is repeated for a selected number of times. Eventually, a list is printed listing the name of the pot and the number of the iteration for each in which the error threshold has been exceeded. Additionally, the relative frequency, mean and variance of all listed errors are printed.

RUNI :: Test of the system interrupts under different AC modes. The number of EFIs is counted which occur in a given time interval and the content of the interrupt cells is checked. Errors are diagnosed.

LOGIC :: Test of sense lines and control lines (via closed loops which have to be patched on the AC's logic box)

IOAR :: Test of the I/O-address register in the interface and the multiplexer and converter address decoding.

LOOP :: Test of the data transfer to and from the AC. The DAC outputs are connected with the multiplexer inputs. Various values are given out via the DACs and received back via the multiplexer—ADC. Any difference between the original value and its received echo which exceeds a preselected error threshold is printed with the statistics of all the trials as in POT.

MULT :: Test of pairs of MDACs. One MDAC is set in increments of .01 between 0 and 1. Its output is multiplied by a second MDAC (which has a constant digital input). Nominal and actual results are compared. Errors are listed, together with all required information.

IDAC :: Test of the interpolating DACs.

It is even less possible to give a general concept of the required LIBRARY PROGRAMS as these depend essentially on the requirements of the users. Programs for parameter optimization routines, function storage and reproduction (with and without delay), fast FOURIER transform, statistical parameter estimation, etc., etc., should be mentioned.

It should also be mentioned that writing the digital part of a simulation program is in our case facilitated by a block oriented digital simulation language which we have developed and which we call SIESTA.[8] SIESTA is a superset of the SCi-CSSL Language,[9] and our SIESTA compiler was one of the first implementations of CSSL.

ACKNOWLEDGMENT

The authors would like to express their grateful appreciation to the Messrs W. Franke (who wrote OPERUN and most of the SUBs), W. Woletz (who wrote HYTROL), and R. Siol (who wrote most of the ACID programs) for their very valuable contribution to the implementation of the system. They are also indebted to Mr. M. E. Connelly, M.I. T. Electronic Systems Laboratory, for his helpful and encouraging suggestions.

REFERENCES

1 W GILOI
Hybride Rechnersysteme
Telefunken-Zeitung Jg 39H 1 1966 82-100

2 W GILOI H SOMMER
PHENO—A new concept of hybrid computing elements
Proc F J C C 1967 23-31

3 W GILOI
Error-corrected operation of hybrid computer
Proc 5th International Conference of AICA Lausanne 1967

4 SCIENTIFIC DATA SYSTEMS
Real-time FORTRAN II manual

5 W WOLETZ
HYTROL—ein Verkehrsprogramm für hybride Rechenanlagen
Diploma Thesis Techn University of Berlin
Inst of Inf Proc 1968

6 W FRANKE
Ein Betriebssystem für hybride Rechenanlagen
Diploma Thesis Techn University of Berlin
Inst of Inf Proc 1969

7 G BEKEY W KARPLUS
Hybrid Computation Chapter 7
J Wiley and Sons Inc New York 1968

8 P RECHENBERG

SIESTA—A CSSL—implementation
Internal Report Techn University of Berlin
Inst of Inf Proc 1968
9 The SCi continuous system simulation language (CSSL)
SIMULATION Vol 9 No 6 December 1967
10 C GREEN H D'HOOP A DEBROUX
APACHE— Breakthrough in Analog Computing
IRE Trans EC-11 October 1962 699–706

A real-time programming language and its processor for digital control of industrial processes

by LIANG LIANG

INTRODUCTION

The acceptance of digital control for large scale industrial processes, such as chemical, refining, power, and material processing is a step towards total automation and a further improvement in control system performance. Process control oriented programming languages provide a control engineer with means to learn and hence set up, modify, and operate a control system with ease. They eliminate the complication of involving a programmer, who generally has little knowledge about process control engineering. The result is a reduction in time and dollars for digital control system generation.

Such a language and processor has been designed and implemented. It has many desirable features: process control engineering orientation, independence of machine, modular structure, convenience of capability expansion, and conversational mode for on-line design and modification. The basic concept to implement a digital control system is borrowed from analog control. The system is created by connecting elementary control modules together. The rationale is that most experienced process control engineers and field-proven control schemes are still heavily analog control oriented. On the other hand, the language can readily be used to implement highly interacting, nonlinear, feedforward, self-tuning types of control. Thus the language serves an immediate need in the process control field while awaiting the breakthrough of large scale multi-control-variable manipulation at the regulatory control level which seems still remote for actual field application.

Specification

Format

A control system is to be implemented modularly. The basic elements are statements and functional blocks. Each block may consist of a number of statements.

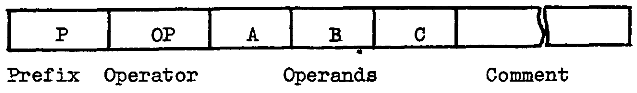


Figure 1—Statement format

The basic format of a statement is a binary operation with an operator and three operands. For example: ADD, A, B, C means $A + B = C$.

The prefix could be the label of the last statement of a loop or a transfer entry. Comments could be added following the operand field. Such a statement format simplifies the syntax of the language and hence the source compilation. Although the format is simple, it is wholly adequate for this application. A variety of operations can be implemented. These include: arithmetic, control and logic functions, I/O handling, matrix, and decision table manipulation. The resulting control system resembles an analog control system with statements replacing analog control modules and blocks of statements replacing cabinets of control modules. Program modularity has further advantages for programming and man-to-man communication, adaptability to change and expansion. Beyond these convenience aspects, this organization makes it possible to implement some important features, such as block execution according to priority (handling emergencies), sequence (process startup), sampling period (process normal control), and time-sharing (to utilize computer free time), on-line design and modification, and block execution time estimation.

Control system decomposition

It is left to the judgment of the control engineer as to how a control system should be decomposed. Consider as an example, a control system is to be designed

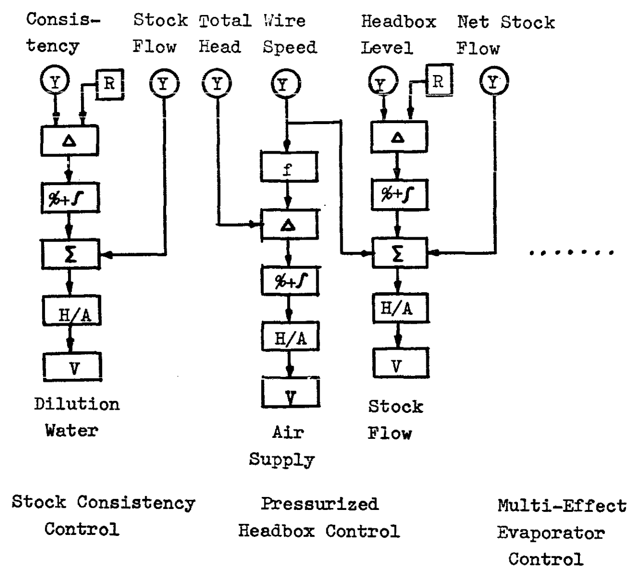


Figure 2—Paper making process control

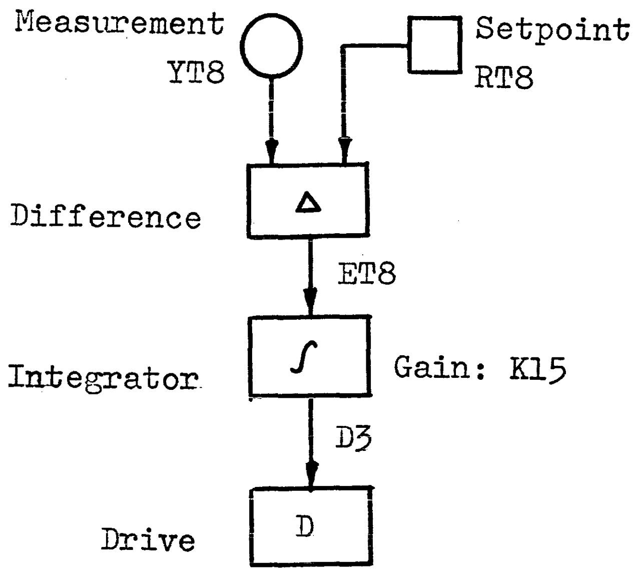


Figure 3—A control loop

for a paper making process. The control engineer may draw up the control schematic as he used to and may divide the control system into blocks according to functions (stock consistency control, pressurized headbox control, multi-effect evaporator control, etc.)

Source coding

A simple example is used to illustrate the coding of a source program. A single control loop is assumed. The control engineer assigns names to variables and parameters and starts to code the source program directly from the schematic.

......	......
a. EBK, PRESS	END OF PRESSURE CONTROL BLOCK
b. DBK, TEMP, 5	TEMPERATURE CONTROL BLOCK
c. DNS, DFS	DECIMAL, FLOATING POINT, SINGLE PREC.
d. SUB, RT8, YT8, ET8	TEMPERATURE ERROR SIGNAL
e. INT, ET8, K15, D3	INTEGRAL CONTROL
f.	......
g. EBK, TEMP	END OF TEMP. CONTROL BLOCK
h.	......

with the interpretation:

a. Declare the end of previous block PRESS;
b. Declare block TEMP to be executed every 5 seconds;
c. Declare number system to be used for the block: decimal, floating point, single precision;
d. Subtract measurement YT8 from setpoint RT8 and obtain error signal ET8;
e. Integrate error signal ET8 with integral gain K15 and generate driving signal D3;
f. More statements may be added;
g. Declare the end of block TEMP; and
h. More blocks may be added.

Rules and procedures

A fixed statement format is used. Functional blocks must be declared. Both transfers and looping must be confined to the same block. Statements may be entered one by one through a teletype or in a lump sum through reading a stack of cards. During source compilation, the block execution time is estimated as a check on the proper assignment of sampling period to the block.

On-line design and modification

During design, check-out, installation or operation phase, changes made on a control system can be anticipated. Thus it is desirable to implement changes on-line, without recompiling the total source program and without affecting the normal running of the process and the control system.

Here, the control engineer refers only to the source program and communicates with the computer in a conversational mode. A version of the source program is kept in the memory and is updated automatically whenever a modification is made. The following on-line modification can be made:

a. Insert a block;
b. Delete a block;
c. Restore a deleted block;
d. Modify block sampling periods;
e. Redefine decimal place of a block (fixed point number);
f. Insert one or more statements;
g. Delete one or more statements;
h. Modify parameter or variable values; and
i. Modify the state of a Boolean variable.

Some typical illustrations are:

```
a. ISB, CONTL, 10;
b. ISS, CONTL, 5, 1
   ADD, SPE, STE, TQ;
c. DLS, CONTL, 3, 2;
d. MSP, CONTL, 5;
e. MPV, K15, 12.5;
```

Their interpretations are:

a. After the teletype is in Modify Mode, the control engineer enters the operation code of inserting a block, ISB; block name CONTL; and block sampling period of 10 seconds. A comma is used to separate items and a semicolon is used to terminate a message. The latter initiates on-line compilation. The new block will immediately be integrated to the operating program and will be executed once every ten seconds;
b. Insert one statement SPE + STE = TQ into block CONTL following the fifth original statement;
c. Delete two statements from block CONTL starting from the third statement;
d. Modify sampling period of block CONTL to five seconds; and
e. Modify the value of parameter K15 to 12.5.

The evaluation of the result of a modification requires human judgment. The control engineer is expected to know what he is doing. A more advanced approach is to let the computer accept or reject a modification.

On-line design of a control system is an application extension of on-line modification. That is, one can practically start from scratch and build up the control system by inserting blocks and statements while the control system is running.

Implementation

The language processor is functionally divided into three major parts: Supervisor, Compiler and Modifier. The Supervisor coordinates and schedules the proper execution of all the programs (language processor, man-machine communication, utility programs and control system). The Compiler translates all source entries into machine language object code. The Modifier realizes on-line modification. The implementation of the Compiler will not be discussed. However, the highlights of the Modifier will be given.

After the Supervisor identifies an input as an on-line modification operation, the Modifier takes over and the proper subroutines are called to process the request. Assume an input ISB, CONTL, 5; is entered. The corresponding source will be:

```
DBK, CONTL, 5   (Declare block)
EBK, CONTL      (END of block)
```

and the object listing becomes (expressed in assembler language for clarity):

```
Q1 TRU Q2              (Transfer)
Q2 TRU SUPERVISOR      (Transfer)
```

where Q1 is the entry point for transfer from the Supervisor. The first object instruction is a transfer to the second instruction, and the second, a transfer back to the Supervisor.

The corresponding data file will be updated (block name, type, sampling period, execution condition, number of statements in the block, starting and ending location of the block and of each statement in the block). The new block will now be executed every five seconds. Assume the statement ADD, A,B,C is to be inserted into block CONTL following the first statement. The source becomes:

```
DBK, CONTL, 5        (Declare block)
ADD, A, B, C         (A + B = C)
EBK, CONTL           (End of block)
```

The new object listing is:

```
Q1 TRU Q3              (Transfer)
Q2 TRU SUPERVISOR      (Transfer)
Q3 LDA A               (Load)
   ADD B               (Add)
   STA C               (Store)
   TRU Q2              (Transfer)
```

where patching method is used. Suppose the statement ADD, A,B,C in block CONTL is to be deleted. The source resumes its original form:

DBK, CONTL, 5	(Declare block)
EBK, CONTL	(End of block)

The object listing becomes:

Q1 TRU Q3	(Transfer)
Q2 TRU SUPERVISOR	(Transfer)
Q3 TRU Q4	(Transfer)
ADD B	(Add)
STA C	(Store)
Q4 TRU Q2	(Transfer)

Thus the deleted statement will be excluded. The implementation of the rest of the modification operations entails little difficulty and will not be discussed. But there are several design considerations worth mentioning. Error messages should be typed out whenever a wrong entry is made. Whether the object codes corresponding to a source statement is a direct insertion or branching to a subroutine, the choice should be an appropriate balance between memory space and execution time. Safety features should be incorporated in the object program, such as saturation on arithmetic operation overflows. The capability of the language should be tailored according to needs. Much of the sophistication can be excluded for many simple applications. Even though only teletype has been mentioned, the use of a CRT or a console with push buttons as man-machine interface is feasible. Only minor modification of the language processor is required.

Experiment

The purpose of the experiment is to demonstrate the use of the language to design a control system on-line. A simplified heat exchange process with two inputs and two outputs was simulated by an analog computer.

The simulated system hardware set-up is shown in Figure 5.

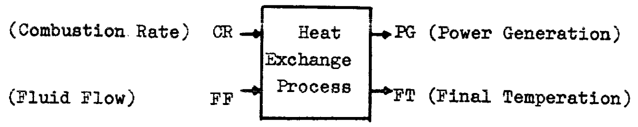


Figure 4—Process model

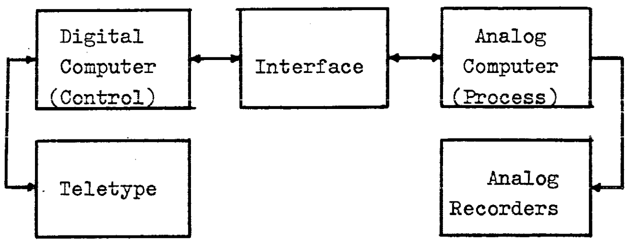


Figure 5—Simulated system hardware set-up

Two single control loops were designed with power generation controlling combustion rate and final temperature controlling fluid flow. The control schematic was prepared as in Figure 6.

The algorithm of the other control loop was the same except with different naming of variables and parameters. While the process was running, the first control block was declared and then statements were inserted one after another until the control loop was constructed. The control gains and set-point were set respectively. The source listing of the first control loop was:

DBK, CTL1, 2	(Declare block)
SBT, IØ, PG	(IØ → PG)
SBT, KØ, RPG	(KØ → RPG)
SUB,RPG,PG,EPG	(RPG − PG = EPG)
SBT, EPG, O2	(EPG → O2)
MUP, K2, EPG, CR1	(K2*EPG = CR1)
INT,EPG,K3,CR2	(∫ K3*EPGdt = CR2)
ADD,CR1,CR2,CR	(CR1 + CR2 = CR)
SBT, CR, OØ	(CR → OØ)
EBK, CTL1	(End of block)

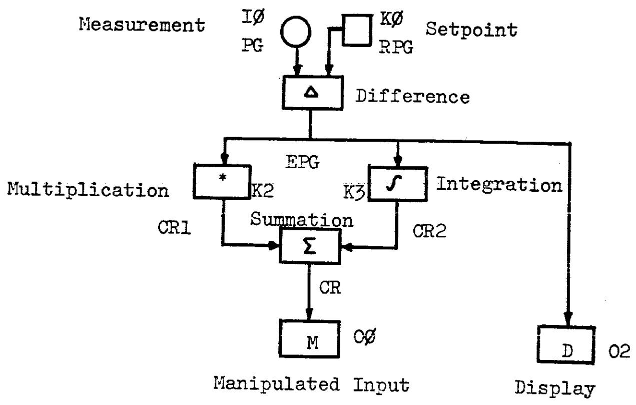
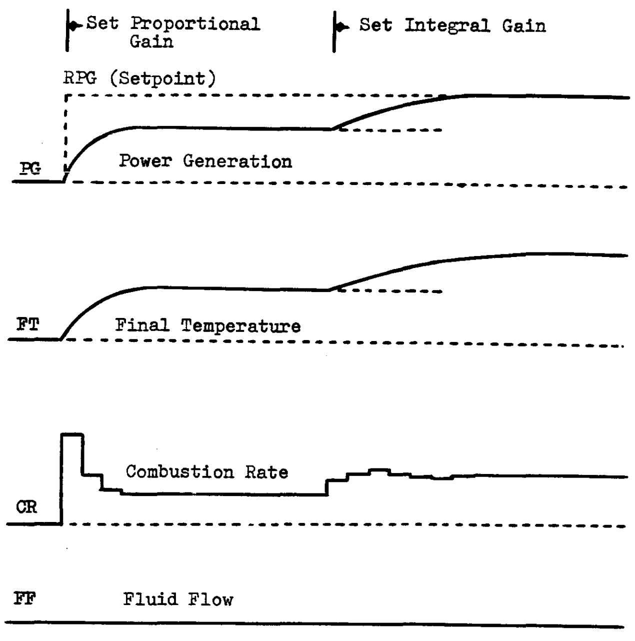


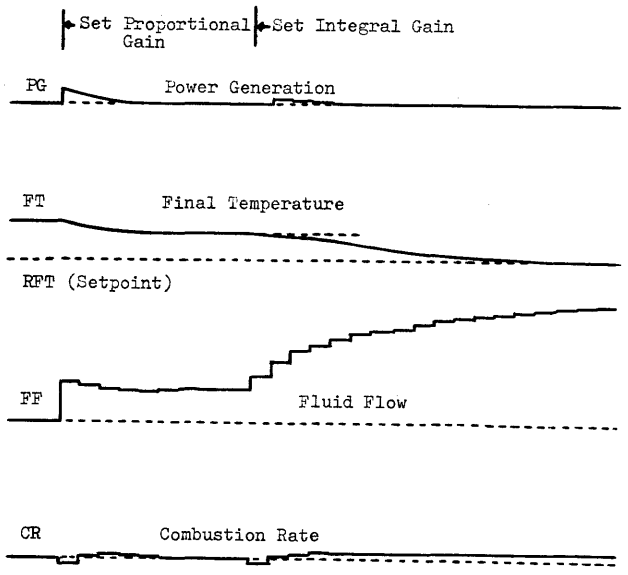
Figure 6—Combustion rate-power generation control loop

In the same way, the second control loop was constructed. The results shown in the following diagrams



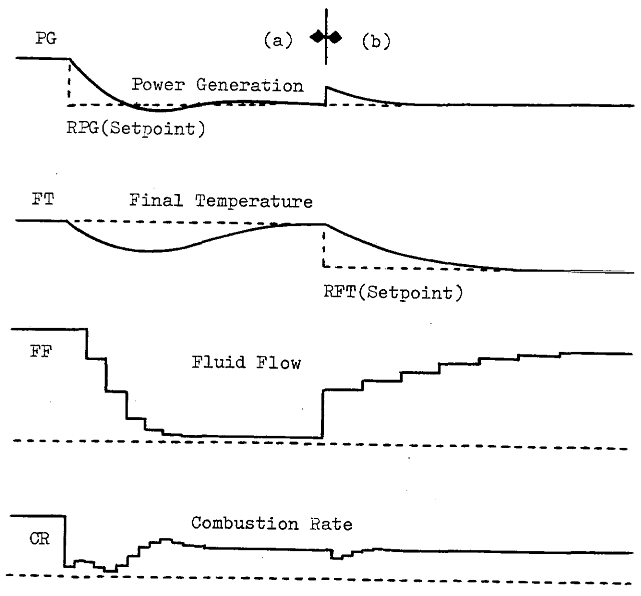
Step Response of Power Generation (PG) by Manipulating Combustion Rate (CR)

Figure 7—Response of process af ter first control loop installed



Control of Final Temperature (FT) by Manipulating Fluid Flow (FF)

Figure 8—Response of process after second control loop installed



(a) Reduce the sampling period of PG-CR control loop by half and make a step change of RPG

(b) Double the sampling period of FT-FF control loop and make a step change of RFT

Figure 9—Response of process by modifying sampling periods of control loops

were highlights chosen from a recorder which was running while the control system was being designed online. It must be noted that they look the same as those obtained by a digital control system designed off-line. However, there is a difference that this control system was gradually built up block by block and statement by statement while the process was running.

CONCLUSION

The language has been recognized by individuals with relevant training, to be quite easy to learn and use. The organization makes both the language processor and the control system flexible for change and expansion. The statement format is simple but adequate. Source compilation is straightforward. On-line design and modification which resembles wire-patching on an analog computer has been demonstrated to be feasible. Although the language is intended for industrial process control, it can be used for process modeling and scientific computation. In addition, it can be integrated into other programming languages for broader application. The language is also effective for laboratory use as in experimenting with digital control schemes.

ACKNOWLEDGMENTS

This paper is extracted from the author's thesis fulfilling the requirement for a MSEE degree (1967) from Case Institute of Technology with support from Bailey Meter Company.

The author is grateful to his thesis advisor, Professor M. J. McCann. He is also gateful to Professors M. D. Mesarovic, I. Lefkowitz and J. D. Schoeffler of Case Institute of Technology, and to Mr. H. T. Hoffman of Bailey Meter Company for their inspiration.

A new graphic display/plotter for small digital computers

by GRANINO A. KORN, STEVENS SIMONS,
RUSSELL STEINBACH, and CLAUDE WIATROWSKI

The University of Arizona
Tucson, Arizona

INTRODUCTION

This report describes a new inexpensive cathode-ray-tube-display and recorder interface designed to provide graphical output for many of the increasingly popular small digital computers in the 12-to 24-bit class. Our original design was developed to produce differential-equation solutions, phase-plane plots, correlation functions, spectra, and amplitude distributions for on-line digital simulation with the 18-bit PDP-9; but the interface logic is flexible enough to serve many other computers and applications.

Our display design does not involve the use of a storage oscilloscope, so that "dynamic" or changing displays are possible. A single "packed' 18- or 16-bit display word sets both X and Y coordinates of a display point, which halves our refresher-memory requirements. As a novel feature, the cathode-ray beam intensity, line-segment (vector) generation, and X-coordinate incrementing can be controlled not only by programmed instructions, but also by special data words corresponding to unused coordinate combinations. Since the display then requires only data words, completely automatic data-channel operation is possible, i.e., the display can be refreshed or changed with the cycle-stealing automatic data channels built into many of the newer small computers with little or no programming. The display will also operate xy (servo) recorders and a four-channel stripchart recorder.

The entire cost of the display interface, when built with Digital Equipment Corporation logic cards, is less than $1,900—exclusive of power supplies and display oscilloscope. The use of DEC cards is convenient for interfacing with the PDP-9, but logic costs could be halved if the interface were assembled with integrated-circuit logic. In this case, the entire display interface would fit on two or three logic cards.

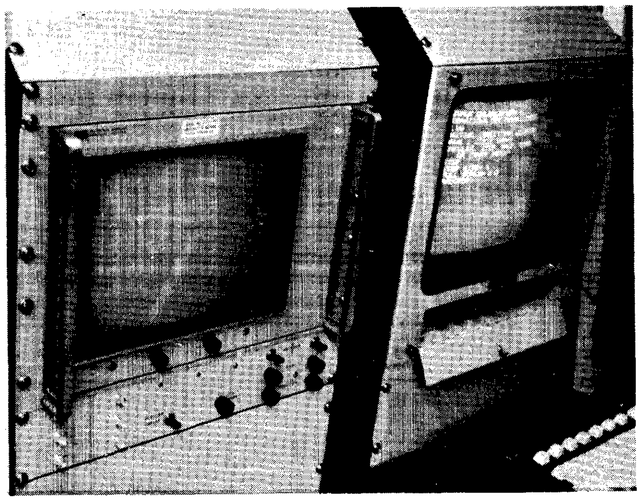
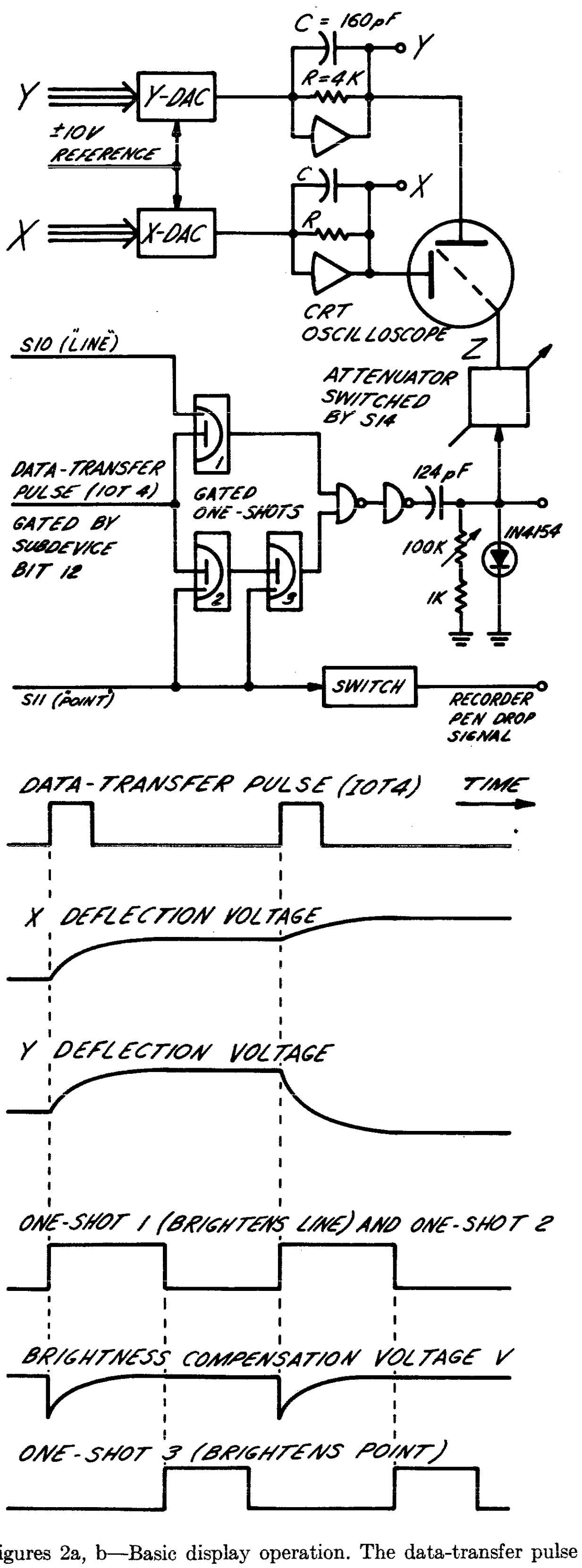


Figure 1—Display unit (left) built into the University of Arizona's DARE (Differential Analyzer REplacement) console. When differential equations or analog-block interconnections are typed onto the CRT typewriter (television monitor at right), the display shows solution curves on-line

Basic display operation

Figure 2 illustrates the design of the digital-computer driven cathode-ray-tube display. Referring to Figure 2a, a point (X, Y) on the oscilloscope screen is positioned through deflection amplifiers driven by 9-bit digital-to-analog converters (X- and Y-DACs). Matched RC delay feed-back networks on the X and Y amplifiers cause both deflection voltages to approach new values exponentially with the same time constant of about 0.7 μsec. (Figure 2b). *When the X- and Y-DAC registers are updated simultaneously, the point (X, Y) then generates a straight-line segment useful for curve interpolation and vector generation,* as suggested by Dertouzos.[1]


C = 160pF
R=4K
Y-DAC
X-DAC
±10V REFERENCE
CRT OSCILLOSCOPE
Z
ATTENUATOR SWITCHED BY S14
S10 ("LINE")
DATA-TRANSFER PULSE (IOT 4)
GATED BY SUBDEVICE BIT 12
GATED ONE-SHOTS
124pF
100K
1K
1N454
SWITCH
S11 (POINT)
RECORDER PEN DROP SIGNAL
DATA-TRANSFER PULSE (IOT4)
TIME
X DEFLECTION VOLTAGE
Y DEFLECTION VOLTAGE
ONE-SHOT 1 (BRIGHTENS LINE) AND ONE-SHOT 2
BRIGHTNESS COMPENSATION VOLTAGE V
ONE-SHOT 3 (BRIGHTENS POINT)

Figures 2a, b—Basic display operation. The data-transfer pulse IOT4 from device selector A or B is gated into one-shots 1 and/or 2 and 3 by the enabling subdevice bit 12 and by S-register bits S10 and S11. DAC's are Pastoriza Electronics Minidac's

Referring again to Figure 2a, the cathode-ray beam intensity is determined by a switched attenuator at the input to the oscilloscope Z-axis amplifier and three gated one-shot multivibrators, which permit *selective brightening of points and/or lines*. These operations are controlled by a 9-bit digital control register, the S register, as follows:

1. S-register bit S14 sets two different beam intensities by shorting a portion of a Z-axis attenuator resistor with a simple FET switch.
2. S-register bit S10 gates the DAC updating pulse into one-shot No. 1 to brighten the screen while a line segment is drawn. As a new feature, *a clamped differentiating network decreases the beam intensity as the beam slows down exponentially* (Figure 2b). This circuit results in more uniform lines than the original Dertouzos circuit.
3. S-register bit S11 enables monostable multivibrators 2 and 3 to brighten the new display point or end point of a line segment (Figure 2b).

Suitable instructions loading the S-register will thus permit the cathode-ray beam to draw points, lines, dash and/or dotted lines, and also to brighten calibration marks on coordinate axes. It is also possible to cause portions of the display to blink.

A primer on program-controlled data transfers

We must next discuss how computer-supplied data words and S-register words are entered into the display registers. As it turns out, a little study of computer-data-transfer techniques will permit us to develop a very versatile set of display instructions.

Practically all computer-interface systems employ a "party-line" I/O bus of the general type illustrated in Figure 3. Here, all DAC and control registers intended to receive data words are permanently wired to the parallel computer I/O data bus. Additional party-line wires carry control-logic signals, which select a specific device and its mode of operation and also synchronize data transmission with the digital-computer operating cycle.

For a minimum of linkage hardware, interfaces work with programmed processor instructions. This method permits great flexibility at the cost of some ingenuity in assembly-language programming. Referring to Figure 3, each input/output (IOT) instruction in the processor instruction register places a specific device-selection code on a set of device selection lines. The *device selector* associated with each individual device is essentially an AND gate which recognizes the device selection-code and gates one, two, or three pulses (IOT pulses) from

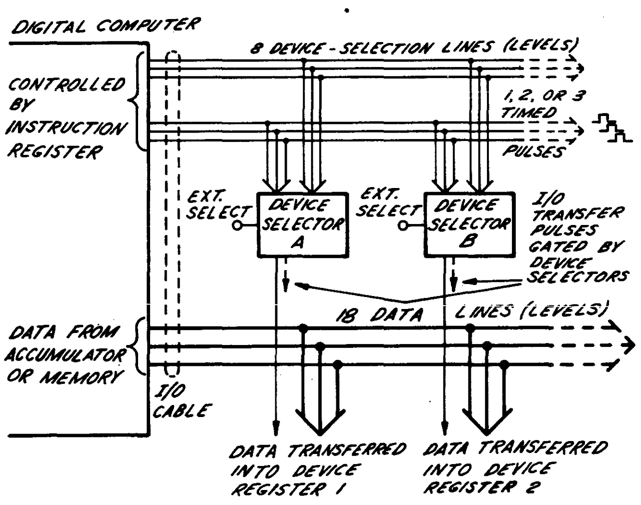

DIGITAL COMPUTER
8 DEVICE-SELECTION LINES (LEVELS)
CONTROLLED BY INSTRUCTION REGISTER
1, 2, OR 3 TIMED PULSES
EXT. SELECT
DEVICE SELECTOR A
EXT. SELECT
DEVICE SELECTOR B
I/O TRANSFER PULSES GATED BY DEVICE SELECTORS
18 DATA LINES (LEVELS)
DATA FROM ACCUMULATOR OR MEMORY
I/O CABLE
DATA TRANSFERRED INTO DEVICE REGISTER 1
DATA TRANSFERRED INTO DEVICE REGISTER 2

Figure 3—Program-controlled operation of a digital computer with party-line I/O bus. An I/O instruction addressed to a specific device is recognized by a device selector, which gates data-transfer pulses to the device in question, as shown in Figure 4

the processor into the selected device to effect data transfers and/or other operations.

Figure 4 shows in more detail how the parallel-connected device-selection and control lines of a typical laboratory computer system (Digital Equipment Corporation PDP-9) correspond to the format of an input/output-transfer (IOT) instruction word in the processor instruction register.[2–5] Bits 0 to 3 of the instruction word inform the processor that an input-output operation is wanted. Bits 6 to 11 place levels (0 or 1) on five *device-selection lines* parallel-connected to all devices on the I/O bus. When these lines carry the *device-selection code* associated with a specific device, its *device selector* (essentially an AND gate, Figure 4) gates (and regenerates) a set of one, two, or three successive processor-timed *command pulses* (IOT pulses) used to effect data transfers and other operations in the selected device in accordance with instruction bits 15 to 17. Not all instructions and devices utilize all three pulses. Bits 12 and 13 of the instruction word ("subdevice bits") can similarly serve to select subdevices or can further gate the command pulses to select device operating modes.

The most common application of the device-selector-gated command pulses is *data transfer* from and to the processor; note that the pulses are synchronized with the processor operation cycle and, thus, with the processor's ability to transmit or accept data. Figure 5 illustrates the principal data-transfer techniques:

1. *Clear-and strobe Transfer* from the I/O bus parallel data lines into the flip-flops of a device register (Figure 5a). Each flip-flop is first cleared by IOT2; then IOT4 strobes the 1's on the data bus into the flip-flop register.
2. *Jam Transfer* (Figure 5b). A single command pulse (IOT4) sets or resets the device-register bits in accordance with the data-bus levels. Jam transfers require slightly more complex electronics than clear-and-strobe, but need only one pulse period for transfer. Jam transfer *must* be used whenever the register resetting opera-

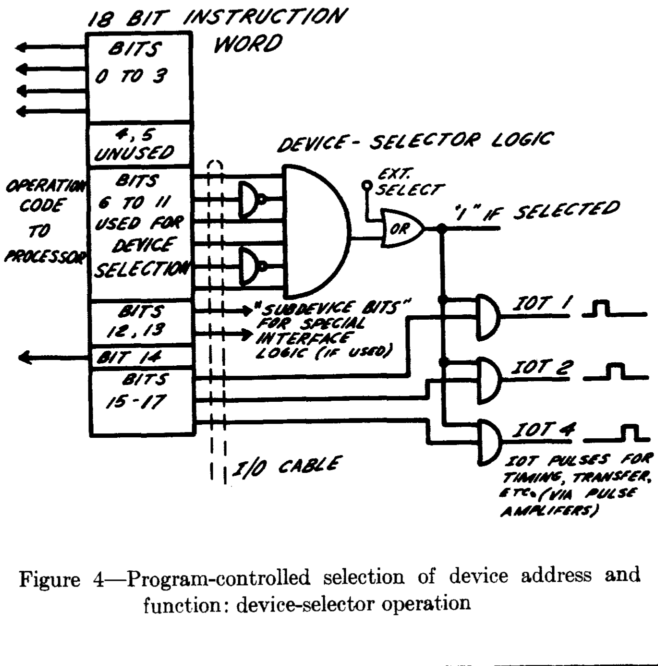
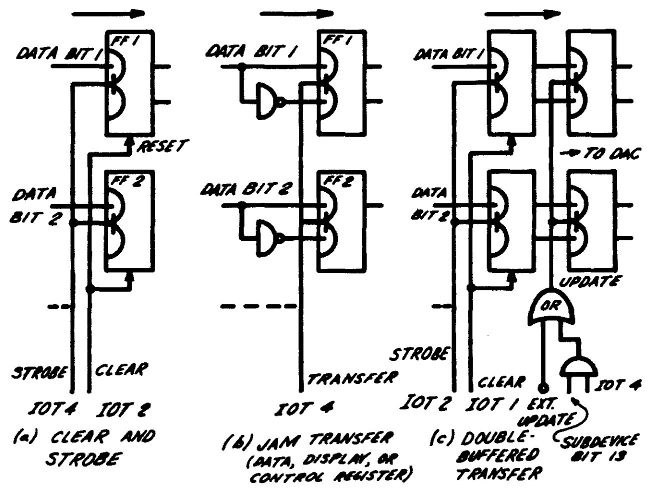

18 BIT INSTRUCTION WORD
BITS 0 TO 3
4, 5 UNUSED
BITS 6 TO 11 USED FOR DEVICE SELECTION
BITS 12, 13
BIT 14
BITS 15-17
OPERATION CODE TO PROCESSOR
DEVICE-SELECTOR LOGIC
EXT. SELECT
OR
"1" IF SELECTED
"SUBDEVICE BITS" FOR SPECIAL INTERFACE LOGIC (IF USED)
IOT 1
IOT 2
IOT 4
I/O CABLE
IOT PULSES FOR TIMING, TRANSFER, ETC. (VIA PULSE AMPLIFIERS)

Figure 4—Program-controlled selection of device address and function: device-selector operation



Figures 5a, b, c—Parallel data transfer: clear-and-strobe (a), jam transfer (b), and transfer into a double-buffered DAC (c). Flip-flops with diode-capacitor gates, like those used in the Digital Equipment Corporation PDP-9, are shown. Level inputs are "0" at 0 volts and "1" at −3 volts, but flip-flop gates set or reset the flip-flop when the level input is "0," and the pulse input goes UP to "0" tion would disturb device functions. This is true, for instance, with DACs required to have a continuous voltage output, and also with control registers which continuously establish a device status.

3. *Double-buffered-register Transfer* (Figure 5c). Data are transferred intothe buffer register by either a clear-and-strobe or jam-transfer operation and are then jam-transferred into the device register. Double-buffered DACs permit simultaneous transfer ("updating") of the analog output of two or more DACs.

As we shall see, suitable device-selecting instructions can also gate IOT pulses into a counter to increment the count.[2]

Display interface and display instructions

Figure 6 shows all the registers of the display interface. The interface employs two separate device selectors, A and B. Each device selector can be addressed by several different display instructions from the processor. Each instruction will generate a different combination of the two subdevice bits and the three IOT pulse obtainable from the device selector addressed, so that many different display instructions can be implemented (Figure 6 and Table I).

Referring to Figure 6, the most important mode of operation involves *simultaneous updating of the 9-bit*

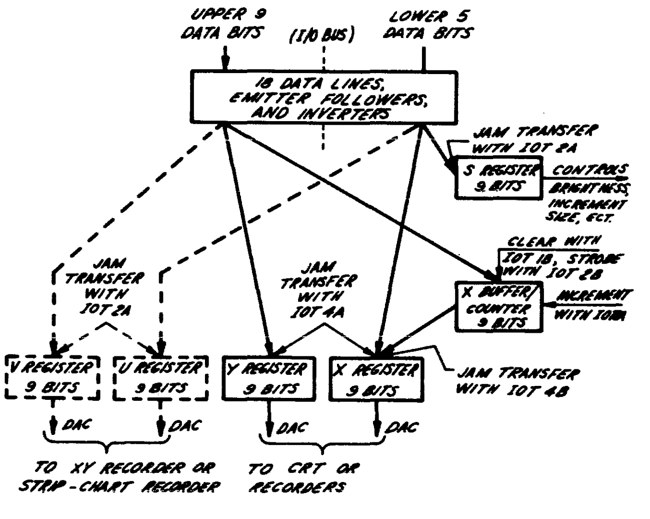


Figure 6—The display registers, with the data-transfer (IOT) pulse effecting data transfers from the I/O bus to the various registers. Subdevice bits can produce many different combinations of the six IOT pulses available from two separate device selectors A and B, so that many different display instructions can be implemented

X-and Y-DACs with "packed" 18-bit PDP-9 words from the I/O data bus:

"Load Y-DAC with data bits 0 to 8, X-DAC with 9-17, and display (i.e., permit either point or line brightening)."

Using octal notation for the instruction-register bits in Figure 3 (each octal digit stands for three bits), the required instruction is

DXYS:	7 0	0 6	4	4
	Code for I/O Instruction	Device Selector A	Subdevice bit 12	IOT 4 (code is sum of IOT pulse numbers)

Such use of a single "packed" 18-bit word for each display point is very advantageous: compared to separate X and Y transfers, we are *halving* data storage requirements, and we are *doubling* the maximum possible point-display rate. If one still desires, however, to load the display with separate X and Y words (say, to avoid the extra time required for word-packing operations when points are displayed in the course of a fast computation), the instruction DXB (Table I) employs device selector B to *clear and strobe the X-buffer*, which is thus loaded with data-bus bits 0-8:

DXB:	7 0	0 5	0	3
	Code for I/O Instruction	Device Selector B	No Subdevice bits are "1"	IOT Pulses 1, 2

(see also Figure 7a). Next, one can *simultaneously load the Y-DAC with data-bus bits 0-8 and the X-DAC from the X-buffer, and display the resulting point or line*:

DYST:	7 0	0 6	6	4
	Code for I/O Instruction	Device Selector A	Subdevice bits 12, 13	IOT Pulse 4

To plot a curve Y(X) with equal X-increments, the X-buffer functions as a *counter* (Figure 7b). The instruction DYSI *increments the X-buffer, then loads Y from the data bus and X from the X-buffer, and displays*:

DYSI:	7 0	0 6	6	5
	Code for I/O Instruct on	Device Selector A	Subdevice bits 12, 13	IOT Pulses 1, 4

Table I lists the most commonly useful display instructions. Figure 7a shows in detail how the IOT

Table I—Display Instructions
(see also Figures 3 and 6)

(*a*) *Basic Instructions*

Mnemonic	Instruction	Octal Code	Device Selector	IOT Pulses	Subdevice Bits
DXYS	Load Y-DAC with data bits 0–8, X-DAC with 9–17, and display	700644	A	4	12
DSL	Load S register with data bits 9–17	700622	A	2	13
DXB	Load X-buffer with data bits 0–8	700503	B	1, 2	—
DYST	Load Y-DAC with data bits 0–8, transfer X-buffer to X-DAC, and display	700664	A	4	12, 13
DXC	Clear X-buffer and X-register	700505	B	1, 4	—
DYSI	Load Y-DAC with data bits 0–8, increment X, and display	700665	A	1, 4	12, 13

(*b*) *Instructions Which Move the Display Point without Brightening*

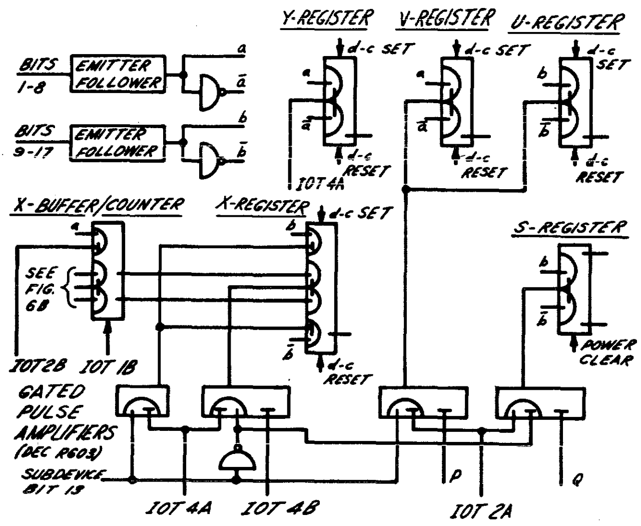
Mnemonic	Instruction	Octal Code	Device Selector	IOT Pulses	Subdevice Bits
DXYL	Load Y with data bits 0–8, X with 9–17	700604	A	4	—
DYLT	Load Y and transfer X-buffer	700624	A	4	13
DYLI	Load Y and increment X	700625	A	1, 4	13
DXL	Load X (Y is unchanged)	700507	B	1, 2, 4	—

(*c*) *Instructions for Fast Drawing of Horizontal Lines*
(Note: Instruction bit 14 clears all data bits in the PDP-9. To draw *vertical* line segments, use DYST.)

Mnemonic	Instruction	Octal Code	Device Selector	IOT Pulses	Subdevice Bits
DXYC	Clear X and Y, and display	700654	A	4	12
DYCT	Clear Y, transfer X-buffer, and display	700674	A	4	12, 13
DYCI	Clear Y, increment X, and display	700675	A	1, 4	12, 13
DXS	Load X with data bits 0–8, and display (Y is unchanged)	700547	B	1, 2, 4	12

(*d*) *Instructions for U and V Channels*

Mnemonic	Instruction	Octal Code	Device Selector	IOT Pulses	Subdevice Bits
DUVL	Load V-DAC with data bits 0–8, U with 9–17	700602	A	2	—
DAL	Load Y and V with data bits 0–8, X and U with 9–17	700606	A	2, 4	—
DAS	Load Y and V with data bits 0–8, X and U with 9–17, and display	700646	A	2, 4	12



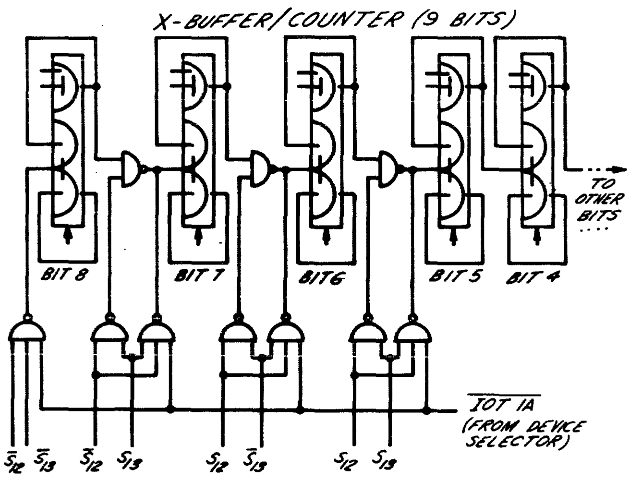


Figure 7a—This figure shows in detail how the different IOT pulses are gated to produce data transfers into the various display registers. Only one flip-flop of each 9-bit register is shown. Calibration switches can set the X, Y, U, and V registers to 0 and −10 V for calibration. The gated pulse amplifiers transmit pulses (−3 V to 0 V to −3 V) when an ungated pulse input is pulsed, or when a gated pulse input is pulsed with the level input at 0 V. Pulse inputs P and Q are for automatic data-channel operation (Figure 8)

Figure 7b—The X-buffer functions as a counter whose increment size is controlled by S-register bits 12 and 13

pulses and subdevice bits corresponding to each instruction gate data into the display registers. Note the following special features:

1. Instructions with subdevice bit 12 = "0" will *move the display point without brightening*. This is useful for creating gaps in lines or curves without any need to reload the intensity-controlling S-register.
2. Instructions clearing either X or Y are useful for *drawing coordinate axes*; selective point brightening produces *scale markers*.
3. Instructions updating Y or X alone can quickly create *vertical and horizontal lines* (coordinate lines, bar charts).

The instruction DSL loads the S-register with data-bus bits 9-17; Table II lists the functions of the individual S-register bits.

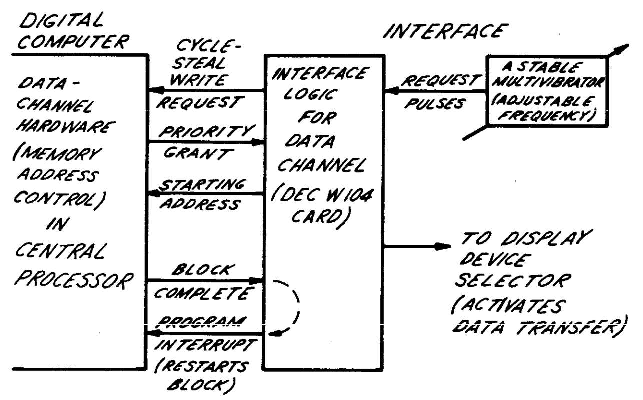
Table II—S-register bit functions

Bit	Function
9	enable data-channel clock
10	display POINT
11	display LINE, or recorder PEN DOWN
12} 13}	X-increment size: 00 = 1 01 = 2 10 = 4 11 = 8
14	Beam intensity
15, 16, 17	Spares

The large number of different instructions possible with only two device selectors has also permitted us to transfer data into two additional DACs (U and V registers in Figure 6), which are intended for use with a four-channel strip-chart recorder but could, in principle, be employed to generate a different display on an additional oscilloscope.

Automatic data-channel operation

While program-controlled display instructions can conveniently alternate with other processor operations (computations and data taking) through program branching and interrupts, much more elegant and efficient display operation may be possible through direct memory access with the *automatic-data-channel hardware* available with many modern small digital computers. Our display is, therefore, designed to work with one of the cycle-stealing data channels built as a standard feature into the PDP-9. Only two processor instructions are needed to specify the *starting address* and *word count* of a block of memory associated with a display picture. The processor then steals memory cycles to output an 18-bit word directly from memory whenever an external *data-request pulse* is sensed. When a complete


DIGITAL COMPUTER
DATA-CHANNEL HARDWARE (MEMORY ADDRESS CONTROL) IN CENTRAL PROCESSOR
CYCLE-STEAL WRITE REQUEST
PRIORITY GRANT
STARTING ADDRESS
BLOCK COMPLETE
PROGRAM INTERRUPT (RESTARTS BLOCK)
INTERFACE
INTERFACE LOGIC FOR DATA CHANNEL (DEC W104 CARD)
REQUEST PULSES
ASTABLE MULTIVIBRATOR (ADJUSTABLE FREQUENCY)
TO DISPLAY DEVICE SELECTOR (ACTIVATES DATA TRANSFER)

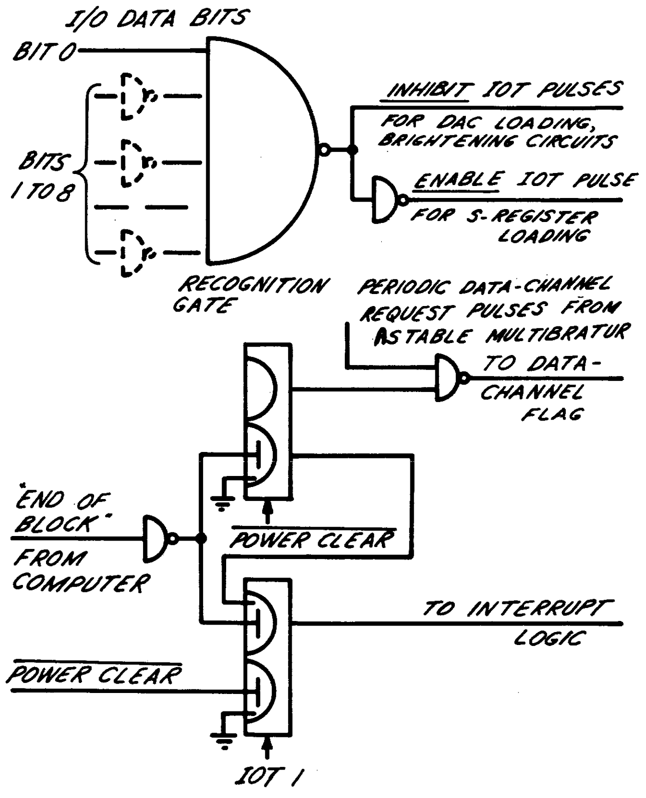

I/O DATA BITS
BIT 0
BITS 1 TO 8
INHIBIT IOT PULSES FOR DAC LOADING, BRIGHTENING CIRCUITS
ENABLE IOT PULSE FOR S-REGISTER LOADING
RECOGNITION GATE
PERIODIC DATA-CHANNEL REQUEST PULSES FROM ASTABLE MULTIVIBRATOR
TO DATA-CHANNEL FLAG
END OF BLOCK FROM COMPUTER
POWER CLEAR
TO INTERRUPT LOGIC
POWER CLEAR
IOT 1

Figure 8a—Automatic-data-channel operation. Processor instructions set the number of words and the starting-word address in a display block and enable the display clock. From then on, data words are transferred to the display on each display-clock request pulse without any need for processor instructions. A data-channel end-of-block signal is used to interrupt the processor, which then restarts the sequence

Figure 8b—Recognition gate for data-channel S-register transfers (see text), and end-of-block interrupt flag

block of, say, 2,000 data words is finished, an *end-of-block pulse* can be used to interrupt the computer program and to reinitiate the block transfer (Figure 8a). No other display instructions are required, and memory locations corresponding to display points can be updated at any time between data-channel outputs. While data-channel word transfer rates as high as 250,000 words per second are possible, we usually set the transfer rate (which is determined by a simple astable multivibrator clock, Figure 8a) at about 100,000 words per second, both to give the computer a chance to compute and to permit cleaner line-segment generation.

On the face of it, data-channel outputs involve only *data words* (in our case packed 18-bit X, Y words, one corresponding to each display point). Since the data channel cannot transmit different display *instructions*, it would seem that we have lost the ability to change between the LINE and POINT display modes and to vary display intensities. If one had more than 18 bits, say with a 24-bit computer, one could use extra data bits as control bits, but this is not possible with the 18-bit PDP-9 and similar small processors. *A special trick, however, permits the use of data-channel produced data words for control purposes.* We recall that, with the 2's-complement code generally employed with DACs, the largest positive 9-bit excursion is 011111111, corresponding to + 9.98 volts, while the largest negative excursion is 100000000 or − 10 volts. This maximum negative excursion, thus, has no positive counterpart and is, in a sense, redundant. *We will, then, employ the last 9 bits of an 18-bit data word of the form 100000000-* sssssssss (Y = − 10 volts) *to set our 9S-bit register bits, rather than for positioning the CRT beam.* Referring to Figure 8b, a simple recognition gate detects the fact that the first 9 bits are 100000000 and gates the remaining bits into the S-register to control the beam intensity and the POINT or LINE mode of subsequent display points.

A second mode of data-channel operation employs the X-buffer/counter *to plot any one-dimensional array automatically against its index, without word packing.*

Operation with external oscilloscopes

To permit the use of our display interface with external cathode-ray oscilloscopes and recorders, the X and Y deflection voltages, an intensity-control voltage, and one of the S-register bits are brought out to front-panel terminals (Figure 2a). In particular, the display can thus operate with a standard five-inch oscilloscope equipped with a Polaroid Land camera to produce quick hard copy. For class demonstrations and visiting admirals, the display also operates with a large 24-inch display oscilloscope at reduced plotting rates. Depending on the external oscilloscope used, the beam-intensifying signal can be amplified and/or inverted.

Operation with xy recorders

To obtain hard copy directly at low cost, one can simply apply the X and Y deflection voltages to an xy recorder (servo table) at a suitably low word rate. For more flexible recorder operation, however, we added two additional digital-to-analog-converter channels (U-and V-DACs and registers, Figure 6).

For servo-table recording, one of the S-register bits is brought out to control the recorder pen-lift solenoid, with the pen DOWN whenever the display instructions call for the LINE mode (Figure 2a). The instruction DUVL produces IOT 2A (Figure 7a) to feed the 9-bit U and V registers with "packed 18-bit words for xy recorder operation. Since the relatively slow servo recorder cannot plot more than 5 points per second, the instructions must be tied to a suitably slow real-time-clock interrupt routine; slow data-channel operation in the manner of Figure 8 is also possible. For the same reason, the 160 pF delay capacitors shown with the X and Y DACs in Figure 2a are replaced with 5 μF plug-in capacitors in the U and V channels, so that line segments are drawn with a time constant of 0.1 sec.

Figure 10 shows a permanent record prepared with the new display unit.

Operation with stripchart recorders

For simultaneous plotting of up to four time-history records on a multichannel strip-chart recorder, the X, Y channels* and the U, V channels are alternately fed packed 18-bit words at a clock-controlled combined maximum rate of 200 words/second. Such multiple time records are especially valuable for recording variables in digital simulation of dynamical systems.

Discussion and follow-on program

Figures 9 and 10 show examples of display operation. As noted earlier, our display was originally intended mainly to produce solution curves and phase-plane diagrams for on-line simulation of dynamical systems. As it turned out, the display is very useful for producing much more general pictures (Figure 9), so that we are planning the addition of a light pen and protractor dial, plus SKETCHPAD-type software[6] for computer-aided drawing operations with the PDP-9.

The most useful feature of our simple display is the possibility of refreshing the display with packed 18-bit words from the standard automatic data channel of the 18-bit PDP-9. Since these 18-bit words fully utilize

* The small capacitors in the X and Y channels (Figure 2a) have essentially no effect at the 50 to 200 Hz word rates used for strip-chart recording.

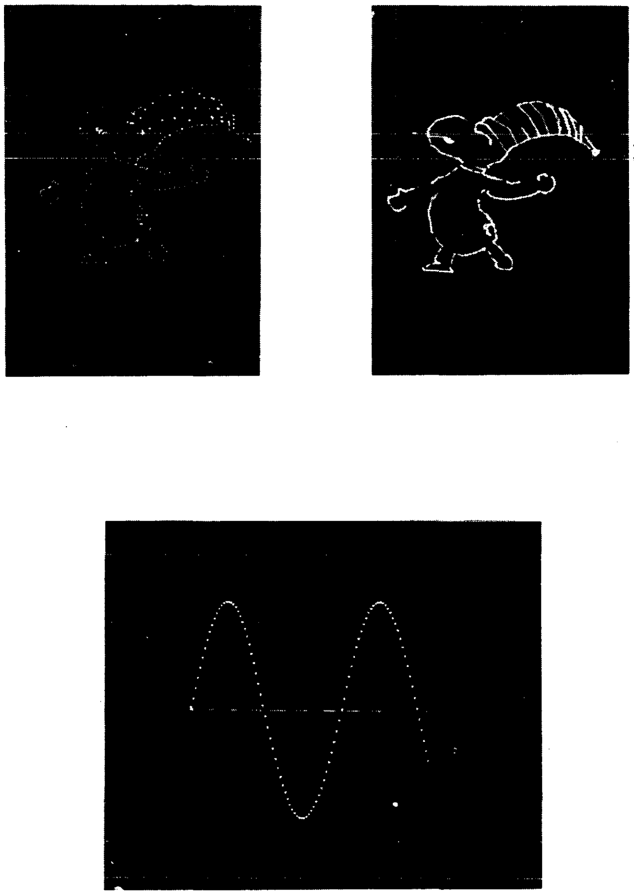


Figure 9—Cathode-ray-tube displays using the PDP-9 data channel at 125,000 points/second: POINT mode (a), LINE mode (b), and mixed POINT and LINE mode (c)

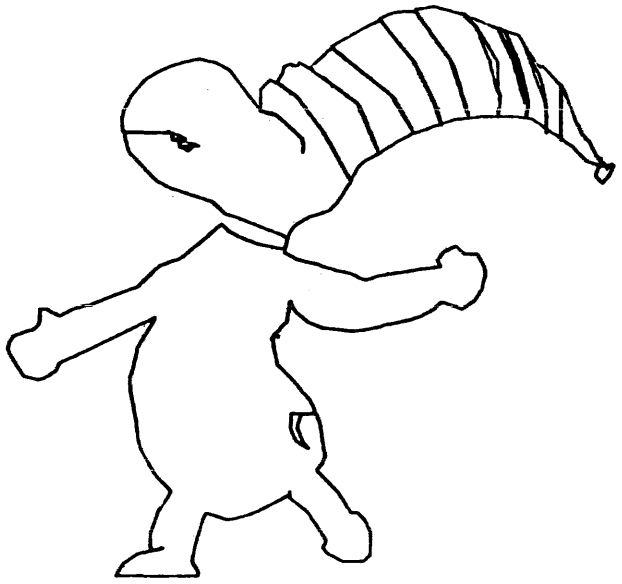


Figure 10—Hard copy produced on a Moseley xy recorder (servo table) in the LINE mode

their PDP-9 registers, we can afford to eliminate the hardware and programming complication of a display-refreshing core or delay-line buffer. With processor and display sharing a common memory, our automatic data channel can do much more than refresh a single display. The channel can also transmit and mix several displays or *parts* of displays (curves, figures, characters), each corresponding to a memory block with a program-selected starting address and block length.[3] Indeed, a block or blocks can be displayed with the automatic data channel while the processor modifies another portion of the display.

The Dertouzos line-segment-generating technique, as modified by our intensity-compensating circuit, was thought to be especially suitable for curve interpolation and stroke-implemented characters. In our actual operating experience, though, most users have employed the LINE mode mainly for servorecorder plots; they seem to prefer the simpler programming possible when using only the POINT mode for CRT display. If we need not alternate LINE and POINT modes it is, for instance, simpler to obtain "incremental" display operation, with successive packed words

$$({}^{k+1}X, {}^{k+1}Y) = ({}^{k}X + {}^{k}\Delta X, {}^{k}Y + {}^{k}\Delta Y)$$

2's-complement-accumulated in the PDP-9 accumulator before transmission to the display or to a data-channel block.

In view of the popularity of small 16-bit digital computers, we have also tried the display with our 9-bit DACs restricted to 8-bit operation. A display thus obtained with packed 16-bit words has, of course, less resolution than the 18-bit display, but still appears to be acceptable for many purposes. This opens the interesting possibility of combining our display system permanently with a small 16-bit processor as a moderately sophisticated stand-alone display unit

The provision of the X-buffer/counter in our display permits *display operation with separate X and Y data transfers*. This is mandatory with 12-bit computers (like the popular PDP-8 series), and even speeds up some PDP-9 programs, since no word-packing is needed (see Appendix A for explicit programmed-data-transfer routines). It is only fair to state, though, that all our users to date have preferred packed-word operation with the PDP-9, so that we could have omitted the entire X-buffer logic (including our extra device selector).

ACKNOWLEDGMENTS

The display system described in this paper is part of a continuing study of digital-, analog-, and hybrid-computer simulation at the University of Arizona. For their support of this study, we are very grateful to the National Aeronautics and Space Administration (Grant NsG-646 and Institutional Grant to the University of Arizona), the National Science Foundation (Grant GK-1860 and Institutional Grant), and to Drs. G. W. Howard, R. H. Mattson, D. L. Patrick, A. B. Weaver, and E. N. Wise of the University of Arizona for allocating institutional-grant funds and University resources.

REFERENCES

1 M L DERTOUZOS
PHASEPLOT: An on-line graphical display technique
IEEETEC April 1967

2 G A KORN
Digital-computer interface systems
Simulation December 1968

3 *PDP-9 user handbook*
Digital Equipment Corporation Maynard Mass 1967

4 *Logic handbook*
Digital Equipment Corp Maynard Mass 1968

5 *Small Computer Handbook*
Digital Equipment Corp Maynard Mass 1968

6 I E SUTHERLAND
SKETCHPAD, a man-machine graphical communication system
Proc S J C C 1963

APPENDIX A: BASIC PDP-9 DISPLAY ROUTINES

```
LAC X      /load accumulator with X                      — 2 μsec
DXB        /load X-buffer with X                         — 4 μsec
LAC Y      /load accumulator with Y                      — 2 μsec
DYST       /transfer X-buffer and Y, and display         — 4 μsec
                                                         --------
                                                          12 μsec

           ------------------------------------
LAC Y      /load accumulator with Y                      — 2 μsec
DYSI       /increment X-buffer, transfer                 — 4 μsec
           /X-buffer and Y, and display                  --------
                                                           6 μsec
```

```
LAW 17000₈          /load accumulator with mask          — 1 μsec
AND Y               /mask last 9 bits of Y out           — 2 μsec
DAC TEMP            /          and save result           — 2 μsec
LAC X               /load accumulator with X             — 2 μsec
CLL                 /clear the link*                     — 1 μsec
LRS 9               /shift 9 places*                     — 6 μsec
ADD TEMP            /combine with Y                      — 2 μsec
DXYS                /transfer X, Y, and display          — 4 μsec
                                                         ————————
                                                          20 μsec
```

```
LAC XY              /deposit X, Y in accumulator         — 2 μsec
DXYS                /transfer X, Y, and display          — 4 μsec
                                                         ————————
                                                           6 μsec
```

* not needed if X was scaled previously

Stability contours for the analysis of analog/ digital hybrid simulation loops

by R.VICHNEVETSKY

Electronic Associates, Inc.
Princeton, New Jersey

INTRODUCTION

The use of hybrid computers for the simulation of dynamic systems has focused interest on the ill-effects of sampling and digital execution time upon solution accuracy. Techniques have been proposed for the compensation of these effects by various authors.[1,3,7] However, there has been a noted lack of common denominator available to compare the relative merits of these methods of compensation. As a rule, one may observe that quality criteria used to evaluate different compensation methods have been local rather than global, and the resulting compensation algorithms are therefore largely tuned to whatever quality criteria has been selected a priori.

As an example, a recent paper by Mitchell[6] shows that compensation techniques that provide an improvement for slightly damped or undamped systems are indeed detrimental when applied to the simulation of highly damped systems. But, at just about the same time, a paper by Deiters and Nomura[1] evaluates the quality of compensation methods on the only merits of their performance in the simulation of completely undamped systems (the circle test).

The preceding remarks indicate that more meaningful criteria should be developed to describe the quality of the compensation over the whole complex plane of natural modes of the system being simulated. Such a criterion is, at least partly, provided by the *Stability Contours method* which is developed in the sequel of this paper. In essence, this method provides regions in the complex plane where the roots or normal modes of the simulated system must lie to ensure stability of the simulation.

An interesting conclusion is reached by consideration of these stability contours; namely, that compensation methods based on Taylor Series extrapolation formulae improve stability for the simulation of undamped or slightly damped systems, but decrease stability for systems which are heavily damped (i.e., have roots which lie closer to the real negative axis than to the imaginary axis).

Reference has been made by previous authors to problems encountered in the simulation of helicopters and STOL aircraft.[6] In that instance, one is faced with a system which is very lightly damped in one regime (during hover or low speed) and becomes heavily damped in another regime (when speed induced aerodynamic forces come into play). From the standpoint of a global analysis such as that provided by the stability contours method developed in this paper, it would seem that none of the compensation methods suggested by previous authors are adequate for the whole range of behavior of the aircraft (unless, of course, the sampling rate is kept high enough to satisfy worst-case stability conditions of the simulation). To the contrary, it seems appropriate to utilize "adaptive.. compensation schemes that will correct by extrapolation when the simulated system is lightly damped, and return to non-extrapolation (or some other algorithm) during flight regimes inducing heavily damped modes of behavior.

A model of hybrid loops

A typical situation encountered in simulation is that in which the differential equations of a dynamic system are integrated on the analog section of a hybrid computer, and part of the non-linear operations involved in the implementation of the right-hand side of these equations is implemented in the digital section of that computer. In particular, this situation is found in the simulation of aircraft where the non-linear aerodynamic forces and moments are computed digitally, and the corresponding integration of the equations of motion is implemented on the analog (see Reference 10).

The sampling and digital delay occurring in the hy-

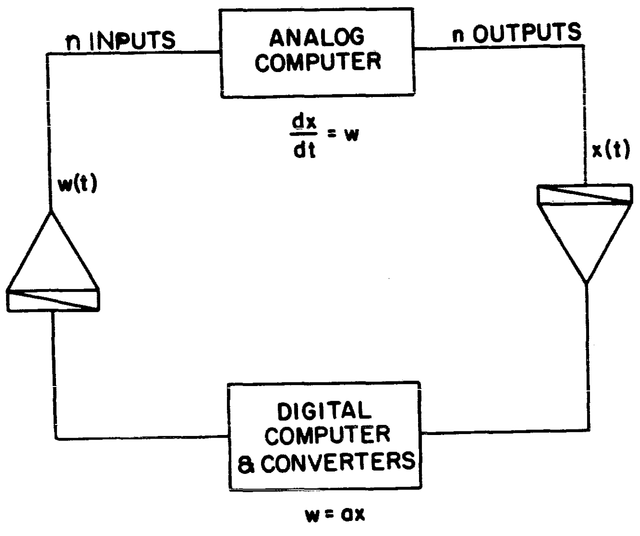


Figure 1—A simplified model of a hybrid simulation loop

brid loop are conducive to instability in the simulation. Although simplified, a model which permits a detailed analysis of this phenomenon is that represented in Figure 1.

This configuration corresponds to the simulation of a system whose set of differential equations is:

$$\left\{\begin{array}{l} \frac{dx_1}{dt} = a_{11} x_1 + a_{12}x_2 + \cdots\cdots + a_{1n}x_n \\ \frac{dx_2}{dt} = a_{21}x_1 + a_{22}x_2 + \cdots\cdots + a_{1n}x_n \\ \vdots \\ \frac{dx_n}{dt} = a_{n1}x_1 + a_{n2}x_2 + \cdots\cdots + a_{nn}x_n \end{array}\right\}$$

or, in simple matrix notations:

$$\frac{dx}{dt} = ax \qquad (2.1)$$

This set of equations may either describe the whole problem at hand, or describe the behavior of error propagations in a more general, non-linear hybrid problem.

In the latter case, a is the jacobian matrix of the problem and is in general time dependent.[9] However, stability of the hybrid loop may be analyzed assuming that a is a constant matrix. This restricts formally the validity of the analysis to the cases where the sampling period Δt is short in comparison to the rate of variation of the coefficients of a. In particular, simulation of VTOL, STOL and variable geometry aircraft where regime or geometry change relatively slowly is well accounted for by the model of Figure 1.

Stability (*the uncompensated case*)

In an idealized implementation of the hybrid simulation of the system described by equations (2.1), we assume that all algebraic computations are performed on the digital computer, while all integrations with respect to time are taking place on the analog.

We also assume that a constant sampling rate Δt is used, and that all analog-to-digital, and digital-to-analog, conversion operations are simultaneous at sampling time (which corresponds to the use of sample/hold integrators in the A/D direction, and double-registers in the D/A direction). We denote by x^i the value of the vector of variables x(t) at time $t = t^i = i\Delta t$.

Using zero-order hold, the output of the digital-to-analog converters are step-like functions. These functions remain constant over one sampling period, and can be expressed by (in the uncompensated case):

$$\{w(t); t \epsilon (t^i, t^{i+1})\} = w^i = ax^{i-k} \qquad (3.1)$$

where x^{i-k} is the value of x at time t^{i-e}.

Thus, $k\Delta t$ is the explicit value of the total sampling and digital execution time.

In view of the previous assumptions, k shall always be an integer, and will often be equal to unity.

During the time interval (t^i, t^{i+1}), the output of the n integrators of the analog computer are linear, and the following expression can be derived:

$$x^{i+1} - x^i = \Delta t \cdot w^i = \Delta t \cdot ax^{i-k} \qquad (3.2)$$

The solution x^i thus satisfies the finite differences equation:

$$x^{i+1} - x^i - \Delta t a x^{i-k} = 0 \qquad (3.3)$$

We apply the z-transformation to equation (3.3) and obtain (I is the unity matrix.):

$$(z - I - \Delta t a z^{-k})\, x^i = 0 \qquad (3.4)$$

(We use the same notation for x^i and its z-transform; the operator z in expression (3.4) is to be interpreted as that which results in the relation: $z^j x^i = x^{i+j}$)

Normal solutions u_j of (3.4) are those which satisfy the equations:

$$u_j^{i+1} - u_j^i - \Delta t \cdot a \cdot u_j^{i-k} = 0 \qquad (3.5)$$

$$u_j^{i+1} = z_j\, u_j^i \tag{3.6}$$

z_j is now a constant scalar, whereas z in (3.4) is a formal operator whose explicit expression need not be defined.

Substitution of (3.6) into (3.5) yields:

$$(z_j I - I - \Delta t \cdot a \cdot z_j^{-k})\, u_j^i = 0 \tag{3.7}$$

Thus z_j must satisfy the characteristic equation:

$$\det(z_j I - I - \Delta t \cdot a \cdot z_j^{-k}) = 0 \tag{3.8}$$

The values of Z_j which are solution of (3.8) and the corresponding u_j, correspond to normal modes of behavior of the simulation. It is well known that, since (3.3) is linear, any x^i can be expanded into a sum of normal modes u_j, and that stability of x^i requires that each of these modes be stable.

Thus, the condition of stability of the simulation is that all the normal modes u_j be stable, which in view of (3.6), is satisfied if all z_j satisfy the relation

$$\left|\frac{u_j^{i+1}}{u_j^i}\right| = |z_j| < 1 \tag{3.9}$$

Formally, (3.9) ensures that

$$u_j^{i+1} = z_j\, u_j^i \tag{3.10}$$

constitutes a contraction mapping,[4] ensuring convergence of (3.5).

Relation to the simulated system's roots

We observe that (3.8) is an algebraic equation of order n(k + 1) in z, and finding all the roots may be a complex task. But the search for the characteristic values z_j can be considerably simplified if a change of coordinates is made so as to diagonalize the matrix a, which corresponds to reducing the original system (2.1) to its normal modes. This is obtained by applying a linear transformation:

$$y = vx \tag{4.1}$$

to equation (3.4), where the lines of v are the eigenvectors of a_ϵ; i.e.,

$$\left.\begin{aligned} a^T v^g &= \lambda_g v^g \\ \det(a^T - \lambda_g I) &= \det(a - \lambda_g I) = 0 \\ g &= 1, 2, \cdots\cdots n \\ v^T &= (v^1\, v^2 \cdots v^g \cdots v^n) \end{aligned}\right\} \tag{4.2}$$

The matrix v has the property to diagonalize a, i.e.,

$$vav^{-1} = \Lambda = \begin{vmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{vmatrix} \tag{4.3}$$

$(\cdot)^T$ stands for the transpose of $(\cdot)$

Writing

$$v(z - I - \Delta t\, az^{-k})\, v^{-1}\, y^i = 0 \tag{4.4}$$

yields

$$(z - I - \Delta t \Lambda z^{-k})\, y^i = 0 \tag{4.5}$$

Since Λ is a diagonal matrix, it follows that (4.5) can be rewritten, line by line, as n independent equations:

$$\left.\begin{aligned} (z_g - 1 - \Delta t\, \lambda_g\, z_g^{-k})\, y_g &= 0 \\ g &= 1, 2, \cdots\cdots n \end{aligned}\right\} \tag{4.6}$$

The significant difference between (3.4) and (4.6) is that the latter is a scalar equation, whereas (3.4) is an n-dimensional equation.

We now look for the normal modes of (4.6), for each λ_g independently, which is a considerably easier task than the original solution of (3.8).

Normal solution of equation (4.6) will be of the form:

$$u_g^{r,i+1} = z_g^r\, u_g^{r,i} \tag{4.7}$$

where z_g^r is one of the (k + 1) solutions of the characteristic equation:

$$z_g^r - 1 - \Delta t \cdot \lambda_g \cdot (z_g^r)^{-k} = 0 \tag{4.8}$$

Stability of the hybrid simulation is expressed by the condition that all z_g^r corresponding to the (k + 1) solutions of (4.8) for each of the n values of λ_g, be in absolute value smaller than unity.

The multiplication of roots

Since equation (4.8) is of the order (k + 1), and has (k + 1) roots, the total number of normal modes of the hybrid simulation is n · (k + 1). Of these, of course, only n are meant to represent the original system; the n · k additional modes are spurious, and their relative amplitude should remain small relative to the original modes to ensure a realistic simulation.

Stability, however, requires that all modes, original or spurious, satisfy the stability criterion:

$$|z_g^r| < 1 \tag{5.1}$$

We assume here, ot course, that the simulated system is itself stable; i.e., that

$$\mathrm{Re}(\lambda_g) < 0$$

$$g = 1, 2, \cdots\cdots n$$

In the next section, we express condition (5.1) graphically by the definition of stability contours in the λ_g complex plane, or plane of the characteristic roots of the simulated system.

Stability contours

In the z complex plane, the condition of stability is that all z_g^r lie inside of the unit circle.

If we transpose this condition into the λ plane, the resulting domain will be that corresponding to systems whose hybrid simulation will be stable. The complex transformation from *Z* plane to the λ plane can be obtained directly from (4.8), which can be rewritten:

$$\lambda_g \Delta t = (z_g^r - 1)(z_g)^k \tag{6.1}$$

A normalization to $\lambda_g \cdot \Delta t$ furthermore provides a dimensionless representation.

We call *Stability Contour* the line, in the $\lambda_g \cdot \Delta t$ plane, which surrounds the domain in which the stability condition (5.1) is satisfied. Along this line, we have:

$$\left.\begin{array}{c} |z| = 1 \text{ or } z = e^{j\omega}; (j = \sqrt{-1}) \\ \omega \epsilon(-\infty, \infty) \end{array}\right\} \tag{6.2}$$

Thus, the equation of the stability contour, obtained by the substitution of (6.2) into (6.1) is:

$$\lambda \cdot \Delta t = (e^{j\omega} - 1)\, e^{jk\omega} \tag{6.3}$$

(For simplicity, we use λ here to denote any of the roots λ_g).

For instance, for k = 0 (no computing delay), this equation reduces to:

$$\lambda \cdot \Delta t = e^{j\omega} - 1 \tag{6.4}$$

which is the equation of a circle of unit radius and center (−1, 0) (Figure 2).

For values of k different from zero, the curve expressed by equation (6.4) presents (k + 1) loops: only that region which is in the inside of all (k + 1) loops corresponds to the stability region. These stability contours, for k = 0, 1, 2, 3 and 4, are shown in Figure 3. For each value of k, the stability domain is that inside of each of the stability contours shown.

It is to be noticed that the stability domain is, as one might expect, decreasing in size for increasing computing delay kΔt.

One may also observe that the crossing point of the stability contour lines with the real negative axis is at distances proportional to 1/(k + 1) from the origin.

We now proceed to evaluate some compensation methods by means of an analysis of their effect upon

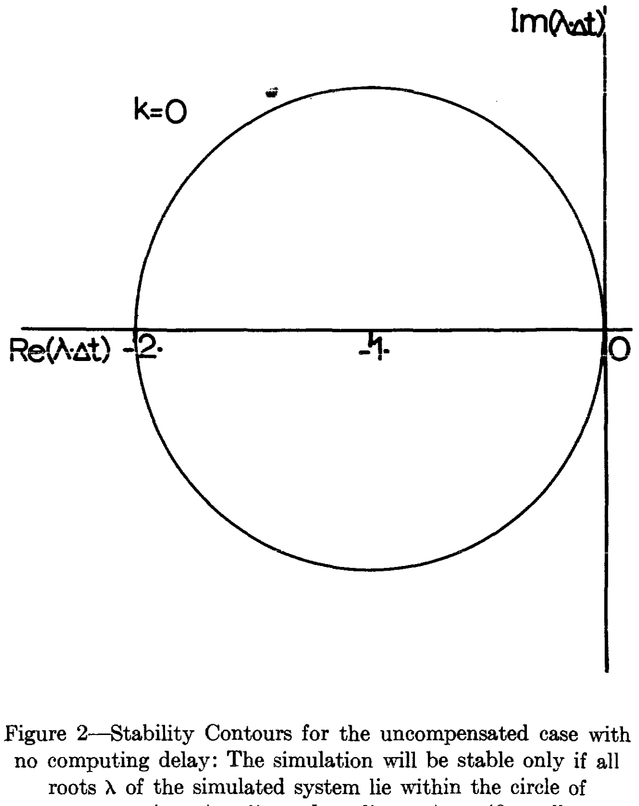


Figure 2—Stability Contours for the uncompensated case with no computing delay: The simulation will be stable only if all roots λ of the simulated system lie within the circle of center (− 1/Δt, 0) and radius 1/Δt. (Or, all normalized roots (λ, Δt) of the simulated system must lie within the circle of center (−1, 0) and unit radius.)

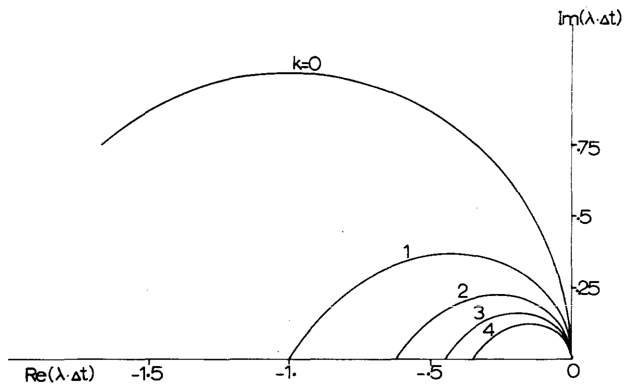


Figure 3—Stability Contours for the uncompensated case—for various computing delays kΔt

stability contours. The compensation methods which we shall analyze are:

1. Compensation by first order digital prediction;
2. Compensation by second order digital prediction;
3. The method of analog compensation.

Compensation by first order digital compensation

The effect of the sampling and digital delay can be alleviated by using digital prediction based on the present and past values of the computed digital computer output w^i. First order prediction is obtained when one past value of w is used in addition to the present value w^i. This allows a linear prediction form to be used. In essence, extrapolating w by $\theta\Delta t$ is given by:

$$\begin{aligned} w^{i+\theta} &\simeq w^i + \theta\,(w^i - w^{i-1}) \\ &= (1+\theta)\,w^i - \theta w^{i-1} \qquad (7.1) \end{aligned}$$

Bearing in mind that $w^i = ax^{i-k}$, this expression becomes:

$$w^{i+\theta} = (1+\theta)\,ax^{i-k} - \theta ax^{i-k-1} \qquad (7.2)$$

$w^{i+\theta}$ is substituted for w^i as the output of the digital computer. Thus, the equation for the computer solution corresponding to the uncompensated case (3.2) is, for this method of compensation:

$$\begin{aligned} x^{i+1} - x^i &= \Delta t\, w^{i+\theta} \\ &= \Delta t\cdot a\cdot[(1+\theta)x^{i-k} - \theta x^{i-k-1}] \qquad (7.3) \end{aligned}$$

or, using the z-operator:

$$\{z - I - \Delta t\cdot a[(1+\theta)z^{-k} - \theta z^{-k-1}]\}\, x^i = 0 \quad (7.4)$$

The characteristic equation (4.8) becomes now:

$$(z-1) - \Delta t\lambda\,[(1+\theta)z^{-k} - \theta z^{-k-1}] = 0 \quad (7.5)$$

The equation defining the stability contour in the $\lambda\cdot\Delta t$ plane is:

$$\left.\begin{aligned} \lambda\Delta t &= \frac{(z-1)z^k}{(1+\theta) - \theta z^{-1}} \\ |z| &= 1 \text{ (or } z = e^{j\omega}) \end{aligned}\right\} \quad (7.6)$$

Stability contours for $k = 1$; $\theta = 0, .5, 1, 1.5$ and 2 are shown Figure 4.

These contours show clearly that while prediction is somewhat improving the correspondence between the stability contour and the axis $\mathrm{Re}(\lambda\Delta t) = 0$, the domain of stability along the real negative axis decreases in some inverse proportion of θ. For the uncompensated case, the stability limit is at $\mathrm{Re}(\lambda\Delta t) = -1$ or one sample per time constant. For the optimum prediction time ($\theta = 1.5$), this stability limit is reduced to $\mathrm{Re}(\lambda\Delta t) = -.483$, or 2.07 samples per time constant.

In the imaginary direction, however, the maximum value of $I_m(\lambda\Delta t)$ on the contour is seen to remain close to .37, in the uncompensated as well as the optimal compensated case ($\theta = 1.5$), which corresponds to 17 samples per pseudo-cycle, although undamped systems ($\mathrm{Re}(\lambda\Delta t) = 0$) always yield unstable hybrid simulations. It will be seen in the next section that second order prediction yields stable hybrid simulations of undamped systems.

Note, however, that a similar effect might be obtained

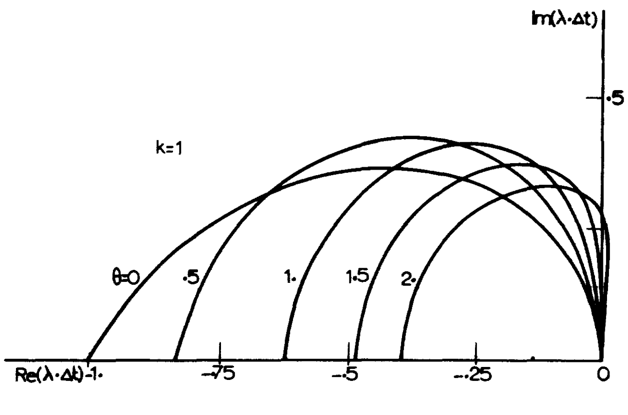


Figure 4—The effect of first order digital compensation upon Stability Contours for k = 1. (The optimum compensation, in the Taylor Series sense, is provided for $\theta = k + 1/2 = 1.5$)

by first order-over prediction. For instance, k = 1, $\theta = 2$ yields, in first order extrapolation, to stable solutions along the imaginary axis up to 22 samples per cycle (lower sampling rates are unstable.)

Compensation by second order digital prediction

Compensation by second order prediction is obtained when two past values of w are used to predict $w^{i+\theta}$:

$$w^{i+\theta} = b_0 w^i + b_1 w^{i-1} + b_2 w^{i-2} \qquad (8.1)$$

The coefficients b_0, b_1 and b_2 are obtained by expressing w(t) as a second order polynomial passing exactly through the three points (t^i, w^i); (t^{i-1}, w^{i-1}); (t^{i-2}, w^{i-2})

This yields:

$$\left.\begin{aligned} b_0 &= 1 + 1.5\theta + .5\theta^2 \\ b_1 &= -2\theta - \theta^2 \\ b_2 &= .5(\theta + \theta^2) \end{aligned}\right\} \qquad (8.2)$$

or

$$w^{i+\theta} = (1 + 1.5\theta + .5\theta^2)\cdot a \cdot x^{i-k} - (2\theta + \theta^2)\cdot a \cdot x^{i-k-1} + .5(\theta + \theta^2)\, a x^{i-k-2} \qquad (8.3)$$

Substituting this expression in replacement of w^i in (3.2), yields

$$x^{i+1} - x^i = \Delta t \cdot a\,[(1 + 1.5\theta + .5\theta^2)x^{i-k} - (2\theta + \theta^2)\, x^{i-k-1} + .5\,(\theta + \theta^2)\, x^{i-k-2}] \qquad (8.4)$$

which, upon making use of the z-operator and the diagonalization of a, yields the equation for the stability contour:

$$\left.\begin{aligned} \lambda\Delta t &= \frac{(z-1)\, z^k}{(1 + 1.5\theta + .5\theta^2) - (2\theta + \theta^2)z^{-1} + .5(\theta + \theta^2)z^{-2}} \\ z &= e^{i\omega} \end{aligned}\right\} \qquad (8.5)$$

Results are shown in Figure 5, for k = 1, θ = 0, .5, 1, 1.5 and 2.

For the optimum value of θ, i.e., $\theta = 1.5$, the crossing point of the stability contour with the real negative axis occurs at $\lambda\Delta t = -.22$, i.e., at 4.5 samples per time constant.

On the other hand, the stability contour for second order extrapolation and $\theta = 1.5$ now crosses the

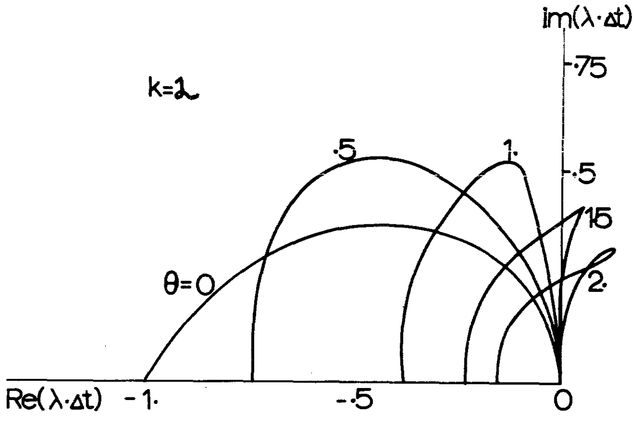


Figure 5—The effect of second order digital compensation upon Stability Contours for k = 1. (The optimum compensation, in the Taylor Series sense, is provided for $\theta = k + 1/2 = 1.5$)

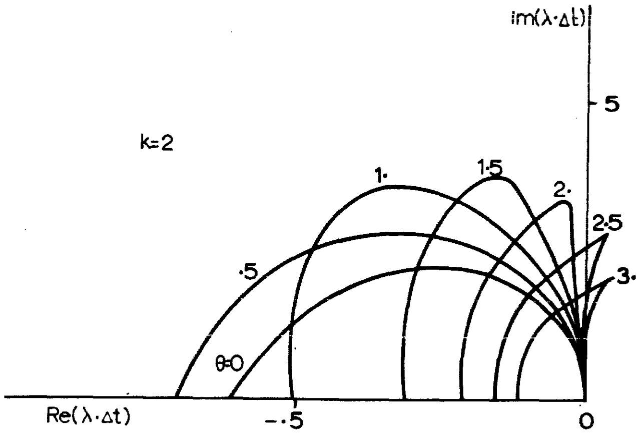


Figure 6—The effect of second order digital compenastion upon Stability Contours for k = 2. The optimum compensation, in the Taylor Series sense, is provided for $\theta = k + 1/2 = 2.5$)

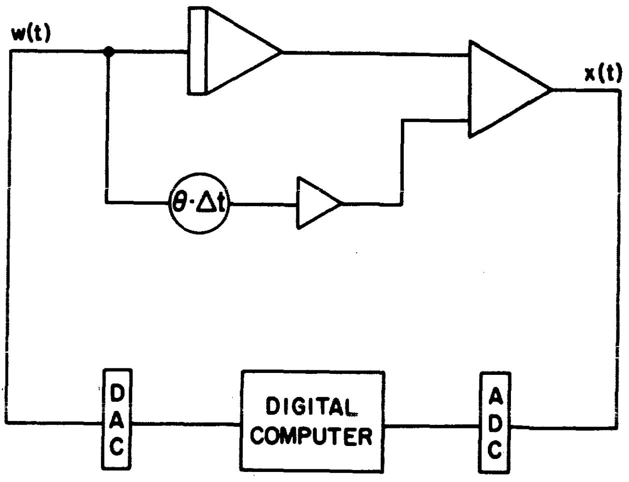


Figure 7—The method of analog compensation of Miura and Iwata

imaginary axis. This occurs for a sampling rate of about 17 samples per cycle.

Figure 6 shows stability contours for second order extrapolation for $k = 2$, $\theta = 0, .5, 1, 1.5, 2, 2.5$ (the ideal extrapolation) and 3.

Analog compensation

A method of analog compensation proposed by Miura and Iwata[7] consists in adding to the output of each integrator $\theta \cdot \Delta t$ times its input (Figure 7). This method and variations thereof were also suggested and analyzed by Karplus.[3]

It can easily be shown that the step equation corresponding to the application of this method is:

$$x^{i+1} - x^{i} = \Delta t \cdot a x^{i-k} + \theta\,(x^{i-k+1} - x^{i-k}) \quad (9.1)$$

or, for the stability contour:

$$\left.\begin{aligned} \lambda\Delta t &= \frac{(z-1)\,z^{k}}{\theta z + (1-\theta)} \\ z &= e^{j\omega} \end{aligned}\right\} \quad (9.2)$$

For

$$\theta = k + \tfrac{1}{2},$$

we obtain:

$$\begin{aligned} \lambda\Delta t &= \frac{(z-1)\,z^{k}}{(k+\frac{1}{2})\,z - (k-\frac{1}{2})} \\ &= \frac{(z-1)\,z^{k-1}}{(k+\frac{1}{2}) - (k-\frac{1}{2})z^{-1}} \end{aligned} \quad (9.3)$$

By comparison with (7.6), we observe that this method of analog compensation results in identical stability contours (and indeed, overall behavior) as those corresponding to cases where k is reduced by 1, and first order ideal ($\theta = k + \frac{1}{2}$) digital extrapolation is applied.

As a means of comparison among the three methods of compensation analyzed, stability contours corresponding to $k = 1$ and $k = 2$ are superimposed in Figure 8 and Figure 9, respectively.

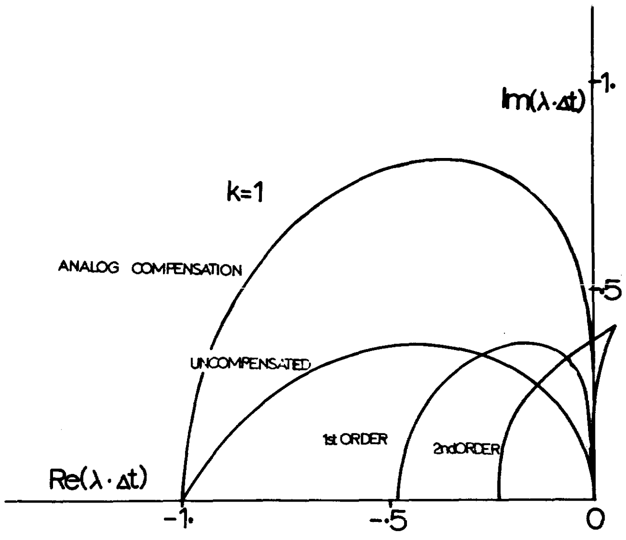


Figure 8—A comparison of different methods of compensation for a computing delay of one sampling period ($k = 1$) and optimal extrapolation (in the Taylor Series sense) $\theta = 1.5$

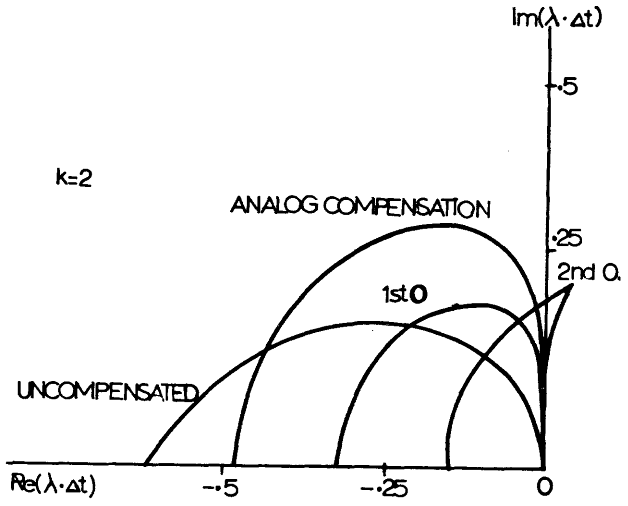


Figure 9—A comparison of different methods of compensation for a computing delay of two sampling periods ($k = 2$) and optimal extrapolation (in the Taylor Series sense) $\theta = 2.5$

CONCLUSIONS

The method of Stability Contours developed in this paper offers a means for analyzing "in the large" the effect of sampling and delay in hybrid loops, as well as a meaningful tool to compare the merits of different methods of digital and analog compensation. It has been applied here to analyze stability properties of compensation methods proposed by previous authors, namely first order digital prediction, second order digital prediction and analog prediction (as well as the uncompensated case).

A general conclusion that appears from such an analysis is that compensation methods based on Taylor Series type of extrapolation algorithms improve the stability properties in the simulation of undamped or slightly damped systems, but are detrimental to stability properties in the simulation of heavily damped systems. This would suggest that, in the simulation of time-varying systems, such as STOL, VTOL and variable geometry airplanes, best results may be obtained by the use of adaptive compensation methods, where the degree of time extrapolation is made inversely proportional to the degree of damping of the simulated system.

REFERENCES

1 R M DEITERS T NOMURA
Circle test evaluation of a method of compensating hybrid computing errors by predicted integral
Simulation January 1967 33

2 R GELMAN
Corrected inputs—A method for improved hybrid simulation
Proc F J C C Vol 24 1963

3 W J KARPLUS
Error analysis of hybrid computer systems
Simulation Vol 6 No 2 February 1966 121

4 A N KOLMOGOROV S V FOMIN
Elements of the theory of functions and functional analysis
Vol I (English translation) Graylock Press Rochester N Y 1957

5 D L MATLOCK
Pulsed prediction filters applied to digital and hybrid simulation
Simulation March 1966

6 E E L MITCHELL
The effect of digital compensation for computation delay in a hybrid loop on the roots of a simulated system
Proc F J C C 1967 Vol 31 103

7 T MIURA J IWATA
Effect of digital execution time in a hybrid computer
Proc F J C C Vol 24 1963

8 A J MONROE
Digital processes for sampled data systems
J Wiley and Sons New York 1962

9 R VICHNEVETSKY
Error analysis in the computer simulation of dynamic systems Variational aspects of the problem
IEEE Transactions on Electronic Computers Vol EC-16 No 4 August 1967 403–411

10 E E L MITCHELL J B MAWSON J BULGER
A generalized hybrid simulation for an aerospace vehicle
IEEE Transactions on Electronic Computers Vol EC-15 No 3 June 1966 304–313

REVIEWERS, PANELISTS, AND SESSION CHAIRMEN

REVIEWERS

E. Alexander
Jonathan Allen
Ramon Alonso
Joel D. Aron
William Atchinson
Algirdaz Aviaienis
Philip R. Bagley
Robert M. Barnett
Frank Bates
Max ben-Aaron
Roger H. Bender
Robert Benenati
Mort Bernstein
Frank F. Bevacqua
Christopher Billings
W. W. Bledsoe
James A. Bloomfield
Jacques Bouvard
Franklin H. Branin
D. B. Brick
J. Reese Brown, Jr.
Roderick R. Brown
Thomas Burke
Peter Calingaert
Duane B. Call
Richard G. Canning
Roy B. Carlson
John J. Carr
John W. Carr, III
William C. Carter
Paul Castleman
T. E. Cheatham, Jr.
J. Chernak
C. K. Chow
W. F. Chow
Carl Christinsen
Yaohan Chu
A. Ben Clymer
Edmund U. Cahler
Martin Cohn
S. F. Condon
Richard W. Conway
F. J. Corbato
W. A. Cornell
Frederick C. Cowburn
Richard Crandall
Robert Daley
L. C. Darden
Fred R. Decker
Richard DeNeufville
Peter J. Denning
Donald L. Dietmeyer
George G. Dodd
R. J. Dowe
Robert Dunn
Joseph Eachus
Jay Early
Murray Eden
Douglas Englebart
William English
Kathryn Erat
Z. W. Esper
Robert Fano
Nick A. Farmer
David J. Farrell
George Fedde
Julian Feldman
Wallace Feurzeig
Tudor Finch
Thomas Fitzgerald
Donald Forina
Franklin H. Fowler, Jr.
M. R. Fox
R. A. Freedman
E. R. Gabrielli
William Gear
E. G. Gilbert
M. E. Gilfix
Russell L. Gilstad
D. E. Goldstein
Joseph W. Goodman
William L. Gordon
A. G. Grace, Jr.
J. N. Gray
Martin Greenfield
Gabriel F. Groner
Thomas G. Hagan
Murray J. Haims
D. Hammel
Patrick J. Hanratty
William Harden
D. R. Haring
George H. Harmon
Frank Heart
Madeline M. Henderson
Gardner C. Hendrie
Robert Hengen
Robert A. Henle
Ernest G. Henrichon, Jr.
Bertram Herzog
Richard H. Hill
D. A. Hodges
Gary Hornbuckle
Austin S. Householder
S. Husson
R. E. Hux
G. T. Jacobi
William Jessiman
Barry J. Karafin
Clarence A. Kemper
Allen Kent
B. Kessel
Charles R. Kime
R. E. King
Arnold L. Knoll
Kenneth Knowlton
Manfred Kochen
Eldo C. Koenig
Walter Koetke
Zvi Kohavi
Jerome M. Kurtzberg
Victor LaBolle
F. W. Lancaster
Eugene L. Lawler
R. W. Lawrence
Donald C. Lincicome
Richard Little
Arthur W. Lo
Jerome Lobel
Eli Lodgenstein
Joseph T. Lundy
D. H. Mack
Richard L. Mandell
Dewey F. Manzer
Michael Marcotty
Harry Martell
E. J. McCluskey
L. P. Meissner
R. Meyers
James C. Miller
Tate Minckler
R. A. Miner
Darrell B. Miskell
E. E. L. Mitchell
Gordon S. Mitchell
George Nagy
Edward A. Nelson
F. William Nesline, Jr.
Nils J. Nilson
Kenneth Olsen

S. Ornstein
Dale Osborn
Jock J. Pariser
Robert W. Parker
F. Perpiglia
H. Philip Peterson
Harold Pickering
A. Pietissanto
Joseph Pistrang
W. R. Plugge
M. Ross Quillin
C. V. Ramamoorthy
Bertram Raphael
Robert Rappaport
William H. Rawlins
Stanley G. Reed
Lawrence C. Roberts
C. A. Rosen
Jack Rosen
Arthur M. Rosenberg
Robert F. Rosin
F. J. Sansom
H. M. Sassenfeld
Harry Scheuer
Max Scholz
Elizabeth Schumacker
Paul Seckendorf
Sally Yeates Sedelow
Oliver G. Selfridge
Herbert Semon
Warren Semon
Peter W. Shantz
Alan C. Shaw
Paul N. Sholtz
R. L. Shuey
John Sidney
R. Silver
Warner Slack
Sanford Smith
F. J. Sparacio
Jason Speyer
Edward P. Stabler
Thomas A. Standish
Thomas B. Steel, Jr.
E. A. Steeves
Einartt Steffend
Mary E. Stevens
Thomas G. Stockham, Jr.
Warren Teitleman
Rankin N. Thompson, Jr.
W. P. Timlake
Larry Travis
Gene A. Vacca
Herbert F. Van Brink
Thomas H. Van Vleck
Robert J. Varney
Robert Vichnevetsky
John V. Wait
Sigurd Wasken
Maria Weller
Frank Westervelt
Charles H. Whelen
Robert R. White
Ronald L. Wigington
James E. Wolle
John H. Worthington
John M. Wright
Kendall R. Wright
Richard K. Yamamoto

PANELISTS

Henriette D. Avram
Milton Bauman
Robert W. Bemer
Edward Bennett
Richard Berman
Adam Block
William F. Brown
James M. Brownlow
G. Edward Bryan
Charles T. Casale
Thomas E. Cheatham, Jr.
Fernando Corbato
John E. Cox
Frank Coyne
John Dearden
Jack B. Dennis
M. L. Dertouzos
D. L. Dietmeyer
Howard W. Dillon
John J. Donovan
Raymond L. Dujack
Kathryn Erat
Alvan R. Feinstein
Wallace Feurzeig
George E. Forsythe
E. R. Gabrielli
Bernard A. Galler
Edward Goldstein
W. L. Gordon
John A. Gosden
Martin Greenberger
Martin Greenfield
T. G. Hagan
Duncan Hansen
William Harden
John A. Harr
James F. Holmes
Grace Murray Hopper
Frank J. Jasinski
W. Jessiman
T. Kallner
Felix Kaufman
Barry Kessler
Eldo C. Koenig
William L. Konigsford
Butler Lambson
William B. Lewis
Andrew J. Lipinski
Milton A. Lipton
W. N. Locke
D. F. Manzer
J. W. Meyer
Tate Minckler
Edward Morenoff
Allen L. Morton, Jr.
Christopher B. Newport
Thomas E. Osborne
Seymour A. Papert
T. G. Paterson
A. J. Pennington
Alan J. Perlis
Owen R. Pinkerman
J. Pistrang
Stanley H. Pitkowsky
Frederick Plugge
O. W. Rechard
Nathaniel Rochester
Robert F. Rosin
Jerome D. Sable
Hal Sackman
Jean E. Sammet
Kirk Sattley
Elizabeth Schumacker
Dan W. Scott
Joseph Seiler
Warner V. Slack
A. R. Solomon
Paul Strassmann
Stuart G. Tucker
John Wright
Lofti Zadeh
William M. Zani
Kenneth L. Zearfoss

SESSION CHAIRMEN

Bruce W. Arden	Robert M. Howe	Richard M. Moroney
Moses M. Berlin	Charles A. Huebner	William H. Ninke
James H. Burrows	Malcolm M. Jones	Elliott I. Organick
Robert N. Davis	Thomas E. Kurtz	George W. Patterson
John J. Donovan	Harry T. Larson	Fred H. Scaife
David Evans	Robert Leonard	Robert H. Stotz
Donald H. Gibson	George H. Mealy	Patrick Suppes
John T. Gilmore, Jr.	Henry S. McDonald	Joseph Sussman
John A. Gosden	James L. McKenney	Leonard Uhr
Martin Graham	Richard E. Merwin	James A. Ward
Robert M. Graham	Richard G. Mills	
Michael A. Harrison	Calvin N. Mooers	

AFIPS PRIZE PAPER AWARD COMMITTEE

C. W. Rosenthal, Chairman	D. R. Heebner	R. J. Preiss
G. E. Bryan	C. A. R. Kagan	
J. A. Githens	M. D. McIlroy	

1969 SJCC LIST OF EXHIBITORS

Access Systems Inc.
Adage, Inc.
Addison-Wesley Publishing Company, Inc.
Addressograph Corporation
Advanced Computer Techniques Corporation
Allen-Babcock Computing Inc.
American Telephone & Telegraph Co.
AMP, Incorporated
Ampex Corporation
Anderson Jacobson Inc.
Applied Data Research, Inc.
Applied Dynamics, Inc.
Applied Magnetics Corporation
Association for Computing Machinery
Astrodata, Inc.
Auerbach Corporation
Auto-trol Corporation

BASF Computron Inc.
Bolt Beranek and Newman Inc.
Boole & Babbage, Inc.
Brogan Associates, Inc.
Bryant Computer Products
Business Information Technology, Inc.

California Computer Products, Inc.
Calma Company
Clevite Corporation, Brush Instruments Div.
Collins Radio Company
Communitype Corporation
Compumatics, Inc.
Computer Applications Incorporated
Computer Automation Inc.
Computer Communications, Inc.
Computer Design Publishing Corp.
Computer Displays Inc.
Computer Methods Corporation
Computer Peripherals Corporation
Computer Products, Inc.
Computer Sciences Corporation
Computer Signal Processors, Inc.
Computer Terminal Corporation
Computer Transceiver Systems, Inc.
Computer Usage Company, Inc.
Computers and Automation
Computerworld
Control Data Corporation

Datacraft Corporation
Data Disc, Inc., Display Division
Data General

Datamark, Inc.
DataMate Computers, Div. of Gamco Industries
Datamation
Data Pathing Incorporated
Data Processing Magazine
Data Products Corporation
Datascan
Data Technology Corporation
Datel Corporation
Delta Data
Di/An Controls, Inc.
Digi-Data Corporation
Digital Development Corporation
Digital Equipment Corporation
Digitronics Corporation
Dura. Division Intercontinental Systems, Inc.
Dynamic System Electronics

Eastman Kodak Company
Edwin Industries
Elbit Computers Ltd.
Electronic Associates Inc.
Electronic Design Magazine
Electronic News, a Fairchild Publication
EMR Computer

Fabri-Tek Incorporated
Factsystem Inc.
Ferroxcube Corporation

General Automation Inc.
General Computers, Inc.
General Design Inc.
General Electric Company,
 Information Systems
  Information Devices Department
  Process Computer Department
  Specialty Control Department
Geo Space Corporation
The Gerber Scientific Instrument Co.

Hendrix Electronics
Hewlett-Packard
HF Image Systems, Inc.
Honeywell, Computer Control Division
Honeywell, EDP
Houston Instrument, Div. of Bausch & Lomb
Hybrid Systems, Inc.

IBM Corporation
IEEE

Indiana General Corporation
Industry Reports, Inc.
Inforex Inc.
Information Control Corporation
Information Displays, Inc.
Information Technology, Inc.
Infotechnics
Interdata Inc.

Jonker Corporation

Kennedy Company
Keydata and Adams Associates Inc.
Kleinschmidt, Division of SCM Corp.

Link Group/Singer-General Precision Inc.
Litton Automated Business Systems
Litton Datalog Division
Lockheed Electronics Co., Data Products Div.

McGraw-Hill Book Company
MAI Equipment Corporation
Magne-Head Div., General Instrument Corp.
Mandate Systems Incorporated
Memory Technology Inc.
Micro Switch, a Division of Honeywell
Milgo Electronic Corporation
3M Company
Modern Data
Mohawk Data Sciences Corp.
Monitor Systems, Inc.
Motorola Instrumentation & Control Inc.

The National Cash Register Company
The National Cash Register Company,
 Industrial Products
Nissei Sangyo Co., Ltd.

Olivetti Underwood Corporation
Omnitec Corp., Div. of Nytronics

Peripheral Equipment Corporation
Photon, Inc.
Potter Instrument Company, Inc.
Prentice-Hall Inc.

RCA Information Systems
RCA Electronic Components
Raytheon Company
Raytheon Computer
Redcor Corporation
Remex Electronics
Rixon Electronics, Inc.

Sanders Associates, Inc.
Sangamo Electric Company
Scientific Control Corporation
Scientific Data Systems
Spartan Books
Stromberg Datagraphics, Inc.
Sycor, Inc.
Sykes Datatronics, Inc.
Systems Engineering Laboratories

Tally Corporation
Tektronix Inc.
Teletype Corporation
Telex, Computer Products Division
Texas Instruments Incorporated
Transistor Electronics Corporation
Tri-Data Corporation
Tyco, Digital Devices Division
Tymshare, Inc.

Ultronic Systems Corp.
 Lenkurt Electric
 General Telephone & Electronics
 Sylvania Electric Products
United Computing Systems, Inc.
UTE (United Telecontrol)
UNIVAC, Division of Sperry Rand Corp.
URS Systems Corporation

Vanguard Data Systems
Varian Data Machines
Vermont Research Corporation
Viatron

Wang Laboratories, Inc.
John Wiley & Sons, Inc.

Xerox Corporation

AUTHOR INDEX